Integrated Master in Chemical Engineering

Particles Swarm Optimization for Chiral Separation by True Moving Bed Chromatography

Master´s Dissertation

By

Joana de Matos Silva

Developed for the Dissertation course unit at:

LSRE-Laboratory of Separation and Reaction Engineering

Supervisor:**Ana Mafalda Ribeiro**

Co-supervisor: **Idelfonso Bessa**

Co-supervisor: **José M. Loureiro**

Chemical Engineering Department

July 2017

“A vida é difícil”

but “We’ll always have Paris”

(Anónimo, Casablanca)

Acknowledgements

An enormous and special THANK YOU to Rui and Mafalda for being daily by my side and for their brilliant ideas which were so important for this work.

I wish to express my gratitude for those who contributed for the completion of this thesis:

Professor José Miguel and Idelfonso for introducing me to the LSRE group and for their supporting and friendship.

Professor Madalena Dias for her advice and just for existing.

304bmates: Albertina for her help with Excel's formatting features and for feeding me with delicious cupcakes ad cookies; Yaidelin for her humor and for her quick and useful solutions for the many things I was too lazy to learn by myself.

To my friends and to the LSRE members who made my days so pleasant and funny.

To the lab. 304b for being my home, during the past months and to my mother for late in the night (very late) driving me to my other home.

P.S. I am very grateful for the existence of *print screen* and *copy-paste*!

Ana Mafalda Ribeiro and José Miguel Loureiro, supervisors of this work, are associated members of the associated laboratory LSRE-LCM funded by FEDER through COMPETE2020 - Programa Operacional Competitividade e Internacionalização (POCI) and by national funds through FCT - Fundaça□o para a Cie^ncia e a Tecnologia.

Abstract

Chiral compounds possess a property called chirality which is characterized by the non equality of the two enantiomers; normally, one of the enantiomers has important therapeutic properties unlike its mirror. The chiral separation assumes, then, a massive importance. Since these compounds are mostly used on the medical field, very high purities are demanded; consequently, using the Simulated Moving Bed (SMB) is a good option to perform those separations.

The chiral separation by SMB is a complex system and studies have been done to optimize its performance. As an alternative, the Particles Swarm Optimization (PSO) is an algorithm that has been developed and, lately, has already been applied to a wide variety of systems.

To simulation purposes, the True Moving Bed (TMB), theoretical model of the SMB, presents several advantages as its model is simpler.

In this work, the separation of the bi-naphthol enantiomers in a TMB device will be optimized, using the PSO algorithm. First, the operating conditions of the TMB device will be optimized and then the dispositive configuration will be added. To this purpose, the objective functions comprise the maximization of the productivity and the minimization of the eluent consumption, respecting a desired purity for the extract and raffinate (this is guaranteed, using penalties).

The PSO is dependent on several parameters whose variance affects the optimization; a study of those parameters (parameters tuning) is then done in this work.

To optimize the operating conditions, three strategies, including different objective functions, are presented: a *Two-steps optimization*, a *Single optimization* and a new variant of the PSO algorithm, the *Parallel* PSO. The *Single optimization* was the one that showed better results when compared with the others in terms of productivity *vs* CPU time. In a general way, all the strategies used in this work led to better results than the published ones for this system.

As the SMB is the dispositive that is used in practice, the equivalence with this model is done and the results showed that the two models approach when the number of columns of the SMB increases.

Results showed that the TMB configuration exerts a serious influence on the separation capacity of the dispositive. In fact, higher productivities were obtained.

Once more, the equivalence with the SMB was done, using the previous results. Moreover, they were used to make a possible configuration to use in the Varicol (a variant of the SMB).

To conclude, an analysis of the algorithm and objective function's behavior is presented.

Keywords: PSO algorithm, SMB model, TMB model, Varicol, chiral separation

Resumo

Os compostos quirais possuem uma propriedade, a quiralidade, que se caracteriza pela inequalidade dos dois enantiómeros; normalmente, um deles possui propriedades terapêuticas, ao contrário do seu espelho. A separação quiral é, por isso, de extrema importância. Dado que estes compostos são essencialmente usados a nível medicial, purezas elevadas são exigidas; consequentemente, o uso do Leito Móvel Simulado (LMS) é uma boa opção para efetuar este tipo de separações.

Uma vez que a separação de compostos quirais pelo LMS é um sistema complexo, estudos têm sido efetuados no sentido de otimizar a mesma. Como alternativa, o algoritmo *Particles Swarm Optimization* (PSO) é um método que, nos últimos anos, já foi aplicado a inumeros sistemas.

Para fins de simulação, o Leito Móvel Verdadeiro (LMV), modelo teórico do LMS, apresenta vantagens, uma vez que o seu modelo é mais simples.

Neste trabalho, a separação dos enantiómeros de bi-naftol em LMV é otimizada, utilizando o PSO. Inicialmente, as condições de operação do dispositivo são otimizadas e, em seguida, a configuração do mesmo é adicionada. Para este fim, as funções objectivo são baseadas na maximização da produtividade e na minimização do consumo de eluente, respeitando uma pureza imposta para o extrato e para o refinado (tal é conseguido, utilizando penalizações).

O PSO é dependente de vários parametros que influenciam a otimização; neste trabalho, um estudo desses parametros é realizado.

Para otimizar as condições de operação do LMV, três estratégias são apresentadas: a *Otimização composta*, a *Otimização simples*, e uma nova versão do PSO original, o PSO *Paralelo*. Os melhores resultados, em termos de produtividade *vs* tempo de computação, foram obtidos, utilizando a *Otimização simples*. De um modo geral, todas as estratégias apresentadas neste trabalho permitiram obter melhores resultados que os existentes na literatura, neste momento.

Uma vez que na prática se utiliza o LMS, a equivalência do LMV com o LMS foi feita; os resultados mostraram que os modelos se aproximam quando o número de colunas do LMS aumenta.

Em relação à configuração do LMV, concluiu-se que exerce uma grande influência na capacidade de separação do equipamento, obtendo-se produtividades mais elevadas.

Uma vez mais, a equivalência com o LMS foi feita e os resultados permitiram ainda elaborar uma possivel configuração para a separação em Varicol (uma variação do LMS).

Para concluir, uma análise do comportamento das funções objetivo e do algoritmo é apresentada.

Palavras-chave: PSO, LMS, LMV, Varicol, Separação quiral

Declaração

Declaro, sob compromisso de honra, que este trabalho é original e que todas as contribuições não originais foram devidamente referenciadas com identificação da fonte.

Porto, Julho de 2017

(Joana de Matos Silva)

Table of contents

Notation and Glossary

a	Column section area	dm^2
A	Less-adsorbed compound	
B	More-adsorbed compound	
$c1$	Acceleration parameter (PSO)	
c2	Acceleration parameter (PSO)	
$c1_0$	Initial acceleration parameter (PSO-TVAC)	
$c1_f$	Final acceleration parameter (PSO-TVAC)	
$c2_0$	Initial acceleration parameter (PSO-TVAC)	
$c2_f$	Final acceleration parameter (PSO-TVAC)	
c	Concentration in the fluid phase	g/L
$d1$	Parameter	
$d2$	Parameter	
Dax	Mass axial dispersion coefficient	dm^2/min
E	Eluent stream	mL/min
EC	Eluent consumption	dL/g
F	Feed stream	mL/min
f	Penalty function	
$factor$	v_{max} coefficient	
$fobj$	Objective funcion	
$fobjE$	Eluent consumption-based objective function	
$fobjP$	Productivity-based objective function	
g_{best}	Best particle	
L	Length	dm
k_L	Mass transfer coefficient	min^{-1}
n_c	Number of components	
N_c	Number of columns	
n_d	Number of dimensions	
n_{it}	Number of iterations	
n_p	Number of particles	
n_s	Number of sections	
P	Purity	
Pe	Peclet number	
$Prod$	Productivity	g/L_{ads}/day
q	Concentration in the solid phase	g/L
q^*	Concentration in the solid phase in equilibrium with the liquid phase	g/L
Q	Volumetric flow-rate	mL/min
Q_s	Solid volumetric flow-rate	mL/min
R	Raffinate stream	mL/min
$Rand(1)$	Random values between 0 and 1	
Rec	Recovery	
t	Time	min
T	Period	
t^*	Switching time	min
u	Fluid interstitial velocity	dm/min
u_E	Eluent velocity	
u_{feed}	Feed velocity	dm/min
u_R	Raffinate velocity	dm/min
u_s	Solid interstitial velocity	dm/min
u_X	Extract velocity	dm/min
v	Particle step	

v_{max}	Maximum particle step	
V_c	Total volume of the column	L
w	Inertia weight	
w_0	Initial inertia weight	
w_f	Final inertia weight	
X	Extract stream	mL/min
x	Particles matrix	
$x_{g_{best}}$	Position of the best particle	
$x_{p_{best}}$	Best position of each particle	
x_{max}	Maximum value for a parameter	
x_{min}	Minimum value for a parameter	
x_p	Particles position	
z	Axial position	dm
Z	Inertia weight coefficient	

Greek letters

ϵ	Bulk porosity
ω	Penalty coefficient
ω_E	Penalty coefficient (eluent consumption)
ω_P	Penalty coefficient (productivity)

Subscripts

I	Section I
II	Section II
III	Section III
IV	Section IV/recycle
A	Less-adsorbed compound
B	More-adsorbed compound
i	Compound in models; iteration in algorithm
j	TMB section
k	SMB column

Superscripts

‘	Respect to SMB

List of Acronyms

CPU	Central Processing Unit
DoE-RSM	Design of Experiments-Response Surface Methodology
FPI	Foreign Process Interface
LDF	Linear Driving Force
LMS	Leito Móvel Simulado
LMV	Leito Móvel Verdadeiro
MUSCOD-II	Multiple Shooting Code for Optimal Control
OCFEM	Orthogonal Collocation in Finite Elements Method
ODE	Ordinary Differential Equation
PC	Prediction-Correction
PDE	Partial Differential Equation
PSO	Particles Swarm Optimization
SMB	Simulated Moving Bed
RANDIW	Random Inertia Weigth
TMB	True Moving Bed
TVAC	Time Variant Acceleration Coefficient
TVIW	Time Variant Inertia Weigth

1 Introduction

1.1 Motivation and relevance

The Particles Swarm Optimization (PSO) is an optimization technique developed by Kennedy and Eberhart in 1995 and is based on the organisms behavior on a social milieu of which a bird flock or a fish school are great examples. The Studies of the synchronic movements of birds by Reynolds (1987) and Heppner (1990) and the studies of the fish schooling by Wilson (1975), all lead to similar conclusions. Besides that, the will to model the human social behavior was also a motivation for the algorithm development (despite the differences between the human and the animal behavior). The statement "In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search of food. This advantage can become decisive, outweighting the disadvantages of competition for food item, whenever the resource is unpredictably distributed in patches" of Wilson (1975) contains the principal idea of the particles swarm: each particle information of the exploitation zone is shared with the other particles which will lead the system, at least in theory, to an optimum.

The PSO algorithm involves simple mathematics and does not demand high computation speeds or memories (Kennedy and Eberhart, 1995). Furthermore, it uses a small number of parameters to adjust and similar parameters can be used for different applications, which makes it so appealing to the optimization of nonlinear functions (Kennedy and Eberhart, 1995; Eberhart and Shi, 2001; Shi and Eberhart, 1998). Nowadays, there exist already some variants of the original PSO (Eberhart and Shi, 2001; Ratnaweera, Halgamuge and Watson, 2004).

The Simulated Moving Bed (SMB) technology is a separating device characterized by working with a countercurrent contact between the solid and the liquid phases. This property makes possible the application of the chromatographic principle to large scale separations. Once the mass-transfer driving force is maximized, a continuous injection is allowed. Although the SMB technology is widely used, its mathematical model is quite complicated; the True Moving Bed is then used to model the SMB.

The True Moving Bed (TMB) model considers that the solid actually moves in the opposite direction of the liquid. The TMB equations are much simpler to use in process simulation since the steady-state can be directly obtained (Rodrigues et al., 2015). In order to find the TMB operating conditions (such as flow-rates) that will lead to the best separation, respecting a desired purity, optimization techniques have been applied. The triangle theory, the DoE-RSM methodology, the prediction-correction (PC) method, algorithms based on the shooting method (MUSCOD-II) and the concept of separation volume are some of the techniques that have already been used (Toumi et al., 2007; Bentley, Sloan and Kawajiri, 2013). The PSO method was applied

to optimize the TMB separation of the bi-naphthol enantiomers by Wu et al. (2006). In this work, a more detailed application of the PSO algorithm to the bi-naphthol enantiomers separation is presented.

The bi-naphthol enantiomers are part of a group called chiral compounds. Chiral compounds are common in the health-related field due to their medical effects. In fact, the two enantiomers have different properties which lead to the possibility of different applications. Chiral separation is, then, very important. Several techniques to perform the chiral separation have been developed such as enantioselective synthesis and diastereoisomeric crystallization (Pais, 1999; Pálovics, Faigl and Fogassy, 2012). In the past few years, the utilization of preparative chiral chromatography has increased due to its high yields and purities of both enantiomers. Following this line, the separation of chiral compounds applied to the bi-naphthol system in a SMB device has been studied by Luís Pais (Pais, 1999; Pais, Loureiro and Rodrigues, 1998).

1.2 Objectives and Layout

The objective of this work is to optimize a TMB unit, using the PSO algorithm, in order to obtain the highest productivity (respecting the purities constraints) to the separation of the bi-naphthol enantiomers.

To this purpose, the text is organized as follows.

A brief explanation of the main concepts involved in this work is made in Chapter 2.

Chapter 3 describes the PSO algorithm and the TMB/SMB mathematical models that are used in this work, along with the bi-naphthol adsorption equilibrium characteristics.

Chapter 4 focus on the computational aspects, including a summary of the algorithm and a block diagram.

The TMB operating conditions optimization, comprising several optimization techniques and objective functions, is reported in Chapter 5. To close this chapter, a comparison of the results is presented.

Using Chapter 5 results, the equivalence with the SMB and its simulation and comparison is made in Chapter 6.

The optimization of the TMB configuration is done in Chapter 7, along with a SMB simulation, considering the obtained configuration. A Varicol case is also proposed.

A detailed analysis of the objective function and the algorithm behavior is made in Chapter 8.

Conclusions and recommendations for future work are proposed in Chapter 9.

2 State of the art

2.1 TMB

Chromatography is a technique that performs the separation of compounds in a mixture through the contact between a mobile and a stationary phase. The use of a stationary phase with different affinities for the compounds in the mixture allows their separation from each other (the compound with less affinity to the stationary phase will “leave” the column first).

Chromatography was first used in the beginning of the nineties. Mikhail Semenovich Tswett (1910), while researching the physicochemical structure of plant chlorophylls, separated the plant pigments (xanthophylls and chlorophylls), using a column of calcium carbonate and carbon disulfide as eluent. The pigments showed different colors as being separated by adsorption thereof, the term *chromatography*, “to write with color”.

David Talbot Day (1897), who observed that a column packed with limestone would change its color through the passage of crude oil, is also one of the contributors to the development of what is nowadays called chromatography.

Since chromatography leads to high purities, it is widely used with pharmaceutic and chemical purposes. Throughout the years, techniques to perform the basic principle of chromatography have been developed and applied to the most diverse applications (Rodrigues et al., 2015).

The TMB concept arises as an application of the continuous countercurrent chromatography in which the solid adsorbent moves in the opposite direction in relation to the liquid phase. A detailed description of the TMB structure and functioning is given in section 3.2.

Although in theory the use of a TMB leads to a great separation due to the maximization of the mass-transfer processes, its application in practice is nearly impossible. In fact, the solid phase would have to actually move which would result in mechanical problems, among others. The experimental application of a TMB is then done by using the Simulated Moving Bed (SMB)(Rodrigues et al., 2015).

2.2 SMB

The Simulated Moving Bed, SMB, was initially developed by Broughton and Gerhold (1961) and Carson and Purse (1962) who proposed the use of a rotary valve (Figure 2.1) which had the function of switching the inlet and outlet streams of the column where the separation was performed.

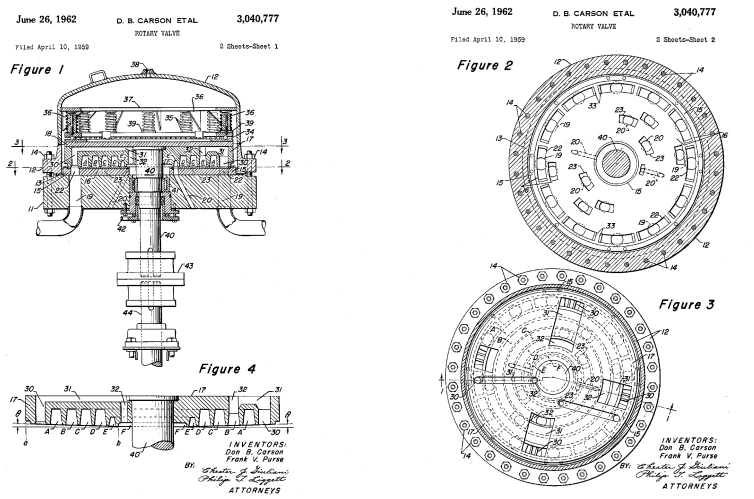

Figure 2.1 Rotary valve drawing by Carson and Purse (1962).

This valve was later replaced by normal valves combined with the use of several columns instead of only one.

Nowadays, the device is constituted by a series of packed bed columns in which the inlet and outlet streams are synchronously switched (using valves) in the direction of the fluid flow. The SMB has two inlets, the feed and the eluent, and two outlets, the extract and the raffinate. At the switching time, t^*, the inlets and outlets change their position cyclically. Figure 2.2 (left) represents the initial configuration and Figure 2.2 (right) shows the new configuration after the first switch (Figure 2.2 considers a SMB device with a column by section). The inlets and outlets continue to change until the initial positions are reestablished (this time is given by $N_c t^*$, in which N_c is the number of columns) (Rodrigues et al., 2015).

In *Modeling strategies for enantiomers separation by SMB chromatography* (Pais, Loureiro and Rodrigues, 1998), the authors show that the approximation of the SMB model with the TMB model improves as the number of columns increases and that usually there is no need of using more than twelve columns.

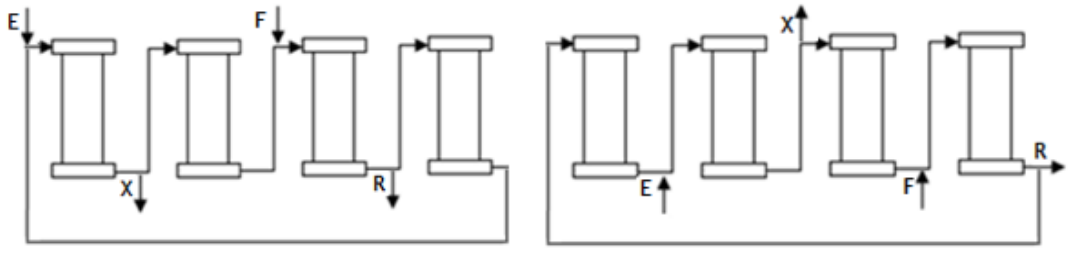

Figure 2.2 Initial inputs and outputs on the SMB device (left); Inputs and outputs after the first switch on the SMB device (right).

2.3 Varicol

The Varicol technology consists on a SMB in which the valves are asynchronously switched; this way, the number of columns per section can be variable if different switiching times are used for the switch of the inlet/outlet streams. This technology advantages are related with the use of a smaller number of columns (comparing with the conventional SMB that would have to be used for a given separation), without compromising the desired productivies and purities.

The Varicol process was developed by *Novasep* and is the non-conventional SMB that is the most frequently used (Faria and Rodrigues, 2015; Gomes et al., 2006).

2.4 Chiral compounds

Some molecules have the characteristic of being mirror images of each other; this property is called chirality. Chiral compounds contain, in general, an asymmetric carbon as a central atom. The mirror images of a chiral compound are called enantiomers. Though the enantiomers are composed with the same chemical material, the fact that the functional groups are in a different position grants them different properties.

Chiral molecules are mostly used for therapeutic purposes such as drugs fabrication (pain relievers and anti-inflammatories) (Pais, 1999). Since that opposite enantiomers can have different pharmacological properties (in some cases one of the enantiomers has no effect or can be harmful (Lorenz et al., 2007)), several techniques to perform the chiral separation have been developed.

2.4.1 Methods for chiral separation

The chiral separation started with Louis Pasteur who, in 1849, separated the sodium-ammonium crystals of racemic tartaric acid (Gamwell, 2016). Since then, many studies were made to develop other techniques to separate enantiomers like crystallization or chromatography. The most common technique to separate enantiomers is chromatography due to its high yields and purities of both enantiomers. However, this process is quite expensive which has led to the development of alternative technologies such as the transformation of the enantiomers in diastereoisomers or the use of optically active solvents. The enantioselective catalysis is another method to separate enantiomers. In this case, only one of the two enantiomers is obtained. If both are desired, two parallel steps are needed (Pais, 1999; Lorenz et al., 2007).

3 Methods

3.1 Particles Swarm Optimization (PSO) algorithm

In the PSO algorithm a system constituted by a family of particles is considered. In the system, each particle keeps track of its coordinates and shares them with the other particles.

The system has three important parameters: the number of particles, n_p, the number of iterations, n_{it}, and the number of dimensions, n_d, (i.e., the number of parameters or variables it is intended to optimize).

The system is initialized with a random family of particles. x represents the particles and corresponds to a matrix with dimensions $n_{it}\times n_p\times n_d$. The position of each particle (i.e., its value), x_p, is given in the first iteration by

$$x_p=x_{min}+rand(1)(x_{max}-x_{min}) \tag{3.1}$$

where x_{min} and x_{max} are the minimum and maximum values for each one of the manipulated variables, respectively, and *rand(1)* is a random number between 0 and 1.

In the PSO method, each particle keeps track of its coordinates which makes possible the determination of the best position of each particle until the current iteration ($x_{p_{best}}$) and the position ($x_{g_{best}}$) of the best of all particles (g_{best}). The values of $x_{p_{best}}$ and $x_{g_{best}}$ are taken into account to move the particles system towards the optimal solution (Eberhart and Shi, 2001).

The variable v defines the particles displacement in the system and will be called *step* in the remainder of this work, for it represents the distance between x_p^i and x_p^{i+1}. In previous works, this variable was referred as "velocity" (Eberhart and Shi, 2001; Kennedy and Eberhart, 1995). In the initial system, the *step* is determined by

$$v=v_{max}(2rand(1)-1) \tag{3.2}$$

here, v_{max}, is given by,

$$v_{max}=\frac{x_{max}-x_{min}}{factor} \tag{3.3}$$

and is an important parameter since it constrains the exploitation zone of a particle. In consequence, v_{max} cannot have a too high value because the particle could miss interesting solutions. On the other hand, it cannot be too low as the particle would barely move and the system would take too much time (iterations) to converge to the optimal solution (Eberhart and Shi, 2001). The *factor* is then a very important parameter since it "decides" how big a particle's *step* can be.

Since the PSO is an iterative method, new values of x_p for each particle dimension are recalculated at each iteration by

$$x_p^{i+1}=x_p^i+v^{i+1} \tag{3.4}$$

where *v* is the *step*.

Note that in previous works (Eberhart and Shi, 2001), x_p was determined by

$$x_p^{i+1}=x_p^i+v^i \tag{3.5}$$

However, once the value of v^{i+1} is already known it will be used in this work.

In the original PSO, Kennedy and Eberhart (1995) proposed that *v* should be update at each iteration by

$$v^{i+1}=v^i+c1rand(1)(x_{p_{best}}{}^i-x_p^i)+c2rand(1)(x_{g_{best}}{}^i-x_p^i) \tag{3.6}$$

where *i* is the iteration, $x_{p_{best}}$ is the best position of each particle, $x_{g_{best}}$ is the position of the best particle, *rand(1)* represents a random value between 0 and 1, *c1* and *c2* are constants.

Later, Shi and Eberhart proposed a new version (1998), the PSO-TVIW (Time Variant Inertia Weight) in which a new parameter, *w* was introduced. The new formula is given by

$$v^{i+1}=wv^i+c1rand(1)(x_{p_{best}}{}^i-x_p^i)+c2rand(1)(x_{g_{best}}{}^i-x_p^i) \tag{3.7}$$

w represents the contribution of the own particle coordinates and can be seen as the "resistance of the particle to its movement" (in previous works *w* was called "inertia weight") and is determined by

$$w=w_0+(w_f-w_0)\frac{i}{n_{it}} \tag{3.8}$$

where w_0 is the inertia weight at the beginning of the search and w_f is the inertia weight at the end of the search (Shi and Eberhart, 1998).

c1 and *c2* are referred in previous works as the "acceleration" and their function is to better balance the contribution of $x_{p_{best}}$ and $x_{g_{best}}$ in the displacement of the particle. In the first works with PSO (Eberhart and Kennedy, 1995; Shi and Eberhart, 1998) *c1* and *c2* were constants. A new version, the PSO-TVAC (Time Variant Acceleration Coefficients) was then proposed by Ratnaweera et al. (2004) in which *c1* and *c2* were calculated by

$$c1=\frac{i}{n_{it}}(c1_f-c1_0)+c1_0 \tag{3.9a}$$

$$c2=\frac{i}{n_{it}}(c2_f-c2_0)+c2_0 \tag{3.9b}$$

in which $c1_f$, $c1_0$, $c2_f$ and $c2_0$ are parameters.

3.2 Steady-State True Moving Bed (TMB)

The True Moving Bed is a separation device in which the solid moves in the opposite direction in relation to the fluid. It has two inlet streams: the feed, *F*, (A+B) and the eluent, *E*, and two outlet streams: the raffinate ,R, (rich in the less-adsorbed compound, A) and the extract, *X*, (rich in the more-adsorbed compound, B). Both solid and fluid are recycled. The TMB unit is divided into four sections (Rodrigues, 2015). In section I, B compound moves with the liquid. In section IV, A moves with the solid. In sections II and III A goes with the liquid and B with the solid. Figure 3.1 represents schematically the TMB, considering a feed with only two compounds, A and B. The green circles represent the nodes.

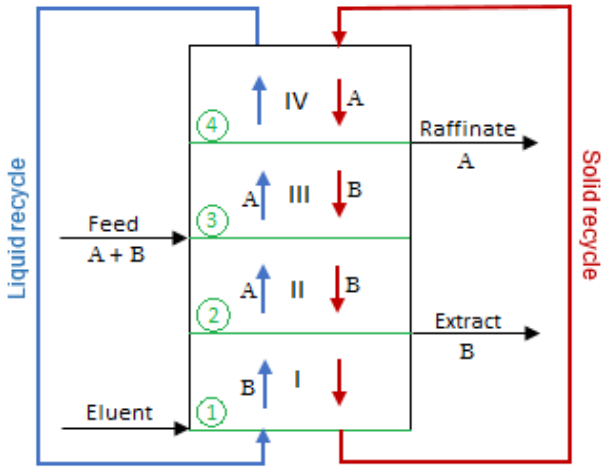

Figure 3.1 Schematic representation of the TMB device.

3.2.1 Mass balances

The TMB mass balances, in steady-state, are presented in the following equations:

- Node balances

$$1)\ u_E+u_{IV}=u_I \tag{3.10}$$

$$2)\ u_I=u_{II}+u_X$$

$$3)\ u_{feed}+u_{II}=u_{III}$$

$$4)\ u_{III}=u_{IV}+u_R$$

where u_j is the fluid interstitial velocity in the section j, u_{feed}, u_R , u_E and u_X are the feed, the raffinate, the eluent and the extract velocities, respectively. Note that the interstitial velocity is given by

$$u=\frac{Q}{a\varepsilon} \tag{3.11}$$

where Q is the volumetric flow-rate, a is the (column) section's area and ε is the bulk porosity.

- Mass balance of component *i* in the fluid phase of section *j*

$$Dax_j \frac{d^2 c_{ij}}{dz^2} - u_j \frac{dc_{ij}}{dz} - \frac{1-\varepsilon}{\varepsilon} k_L \left(q_{ij}^* - q_{ij} \right) = 0 \tag{3.12}$$

Here, *z* represents the axial position, Dax_j is the axial dispersion coefficient in section *j*, c_{ij} is the concentration of compound *i* (in section *j*) in the liquid phase, u_j is the fluid interstitial velocity in section *j*, k_L is the mass transfer coefficient (considering a Linear Driving Force, LDF, aproximation), q_{ij}^* is the concentration of compound *i* (in section *j*) in the solid phase in equilibrium with the liquid phase and q_{ij} is the concentration of compound *i* (in section *j*) in the solid phase.

- Mass balance of component i in the solid phase of section *j*

$$u_s \frac{dq_{ij}}{dz} + k_L \left(q_{ij}^* - q_{ij} \right) = 0 \tag{3.13}$$

where u_s is the solid interstitial velocity.

In the previous balances, the axial dispersion coefficient in section *j* is given by

$$Dax_j = \frac{u_j L_j}{Pe} \tag{3.14}$$

where L_j is the length of section *j* and *Pe* is the Peclet number.

3.2.2 Boundary conditions

At section *j*, the boundary conditions for compound *i* in the liquid phase are given by

$$\text{At } z=0,\ c_{ij} - \frac{Dax_j}{u_j} \frac{dc_{ij}}{dz} = c_{ij,0} \tag{3.15a}$$

$$\text{At } z=L_j,\ 1)\ c_{iIV,L} = \frac{u_I}{u_{IV}} c_{iI,0} \tag{3.15b}$$

$$2)\ c_{iI,L} = c_{iII,0}$$

$$3)\ c_{iII,L} = \frac{u_{III}}{u_{II}} c_{iIII,0} - \frac{u_{feed}}{u_{II}} c_i^{feed}$$

$$4)\ c_{iIII,L} = c_{iIV,0}$$

The boundary conditions for compound *i* in the solid phase are expressed by

$$1)\ q_{iIv,L} = q_{Ii,0} \tag{3.16}$$

$$2)\ q_{iI,L} = q_{iII,0}$$

$$3)\ q_{iII,L} = q_{iIII,0}$$

$$4)\ q_{iIII,L} = q_{iIV,0}$$

3.3 Simulated Moving Bed (SMB)

3.3.1 Mass balances

The SMB balances are presented in the following equations:

- Global balances

$$1)\ u_E+u_{IV}'=u_I' \qquad (3.17)$$

$$2)\ u_I'=u_{II}'+u_X$$

$$3)\ u_{feed}+u_{II}'=u_{III}'$$

$$4)\ u_{III}'=u_{IV}'+u_R$$

where u_k' are the fluid interstitial velocities in the column k, u_{feed}, u_R , u_E and u_X are the feed, the raffinate, the eluent and the extract velocities, respectively. Note that the intersitical velocity is given by

$$u=\frac{Q}{a\varepsilon} \qquad (3.18)$$

where Q is the volumetric flow-rate, a is the (column) section's area and ε is the bulk porosity.

- Mass balance of compound i in the fluid phase

$$Dax_k'\frac{\partial^2 c_{ik}}{\partial z^2}-u_k'\frac{\partial c_{ik}}{\partial z}-\frac{1-\varepsilon}{\varepsilon}k_L\left(q_{ik}^*-q_{ik}\right)=\frac{\partial c_{ik}}{\partial t} \qquad (3.19)$$

Here, z represents the axial position and t represents the operation time, Dax_k' is the axial dispersion coefficient in column k, c_{ik} is concentration of compound i (in column k) in the liquid phase, u_k' is the fluid interstitial velocity in column k, ε is the bulk porosity, k_L is the mass transfer coefficient (assuming LDF aproximation), q_{ik}^* is concentration of compound i (in column k) in the solid phase in equilibrium with the liquid phase and q_{ik} is concentration of compound i (in column k) in the solid phase.

- Mass balance of compound i in the solid phase

$$k_L\left(q_{ik}^*-q_{ik}\right)=\frac{dq_{ik}}{dt} \qquad (3.20)$$

In the previous balances, the axial dispersion coefficient in column k is given by

$$Dax_k'=\frac{u_k'L_k'}{Pe} \qquad (3.21)$$

where L_k' is the length of column k and Pe is the Peclet number.

3.3.2 Initial conditions

The initial conditions are expressed by

$$c_{ik}=q_{ik}=0 \text{ at } t=0 \tag{3.22}$$

Here t represents the time.

3.3.3 Boundary conditions

For column k, the boundary conditions at $z=0$ are given by

$$c_{ik}-\frac{Dax_{k}'}{u_{k}'}\frac{\partial c_{ik}}{\partial z}=c_{ik,0} \tag{3.23}$$

While at $z=L_k'$, they are

$$\text{for the eluent node, } c_{ik}=\frac{u_{I}'}{u_{IV}'}c_{i(k+1),0} \tag{3.24}$$

$$\text{for the feed node, } c_{ik}=\frac{u_{III}'}{u_{II}'}c_{i(k+1),0}-\frac{u_{feed}}{u_{II}'}c_{i}^{feed}$$

$$\text{for the extract and raffinate nodes, } c_{ik}=c_{i(k+1),0}$$

3.4 Equivalence between the TMB and the SMB

The equivalence between the TMB and the SMB in terms of interstitial velocities (by keeping the liquid velocity constant in relation to the solid velocity) or flow-rates, respectively, can be expressed by

$$u_{j}^{SMB}=u_{j}^{TMB}+u_{s} \tag{3.25}$$

$$Q_{j}^{SMB}=Q_{j}^{TMB}+\frac{\varepsilon}{1-\varepsilon}Q_{s} \tag{3.26}$$

Here, Q_s and the switching time ,t^*, are respectively given by

$$Q_{s}=\frac{1-\varepsilon}{t^*}V_{c}' \tag{3.27}$$

$$t^*=\frac{L'}{u_s} \tag{3.28}$$

where, L' is the length of the SMB column and V_c' is the volume of the SMB column (Rodrigues, 2015; Pais, 1999).

3.5 Bi-naphthol adsorption equilibrium

3.5.1 Bi-naphthol

The bi-naphthol ($C_{20}H_{14}O_2$), also known as 1,1'-Bi-2-naphthol, is an organic compound that possesses axial chirality, originating two enantiomers. When purified, the enantiomers can be used as chiral building blocks in asymmetric synthesis and as catalysts in some chemical reactions.

The properties of the bi-naphthol enantiomers (Pais, 1999) are listed in Table 3.1.

Table 3.1 Bi-naphthol enantiomers properties.

Molecular weight, g/mol	286.33
Melting point, °C	215
Solubility limit*, g/L	7-8

*in 72/28 v/v heptane/isopropanol solvent

3.5.2 Stationary phase

To simulate the separation of the bi-naphthol enantiomers, a Pirkle type stationary phase, the 3,5-dinitrobenzoyl phenylglycine covalently bonded to silica gel (3,5-DNBPG-Silica) was used. The particles had a diameter of 25-40 μm (Pais, 1999).

3.5.3 Eluent

The eluent was a 72/28 heptane/isopropanol mixture. The eluent characteristics (Pais, 1999) are in Table 3.2.

Table 3.2 Eluent properties.

	Heptane (C_7H_{16})	Isopropanol (C_3H_8O)
Molecular weight, g/mol	100.21	60.10
Melting point, °C	-91	-89.5
Boiling point, °C	98	82.4
Viscosity at 25 °C, cP	0.386	1.988

3.5.4 Adsorption equilibrium isotherms

The adsorption equilibrium isotherms were determined by the *Separex* (now, *Novasep*) group (Pais, 1999) and are respectively given by

$$q_A^* = \frac{2.69 c_A}{1+0.0336 c_A+0.0466 c_B} + \frac{0.10 c_A}{1+c_A+3 c_B} \tag{3.29}$$

$$q_B^* = \frac{3.73 c_B}{1+0.0336 c_A+0.0466 c_B} + \frac{0.30 c_B}{1+c_A+3 c_B} \tag{3.20}$$

Here, q_i^* and c_i are given in g/L.

4 Computational Aspects

The PSO algorithm and its variants along with the objective functions that were used in this work were written in MATLAB. The TMB, SMB and VARICOL models were written in gPROMS. The Orthogonal Collocation in Finite Elements Method (OCFEM) with second order polynomials in a grid of uniform intervals (the number of intervals was variable depending on the simulation) was used to discretize the ODEs and PDEs. The communication between gPROMS and MATLAB was done with gO:MATLAB, using a FPI (Foreign Process Interface) event.

The simulations were run in a processor Intel® Core™ i5-2400 with a 3.10 GHz CPU. The RAM had an 8.00 GB capacity.

A listing of the computer codes is given in Appendixes A and B. The PSO codes along with the called programs (objective functions and TMB model) are listed in Appendix A. Appendix B presents the listings of the SMB, TMB and Varicol models.

4.1 Simulation Algorithm

The PSO algorithm used in this work is summarized as follows.

1. Set limits (x_{max} and x_{min}) and parameters (n_p, n_d, n_{it}).

2. Calculate v_{max}.

3. Initialize the system (x_p and v).

4. Calculate w, $c1$ and $c2$.

5. Check whether the point is physically possible or not.

6. Evaluate the objective function for each value of x_p.

7. Select $x_{p_{best}}$ and $x_{g_{best}}$.

8. Update x_p and v.

9. Loop from point 5 until the maximum number of iterations is attained.

4.2 Block Diagram

Figure 4.1 shows a block diagram with the computer code structure of the optimization process. In order to make the reading of this flow-chart easier, some of the variables and equations that are used in their calculation are included in Figure 4.1. A description of the variables is given in Appendix A.

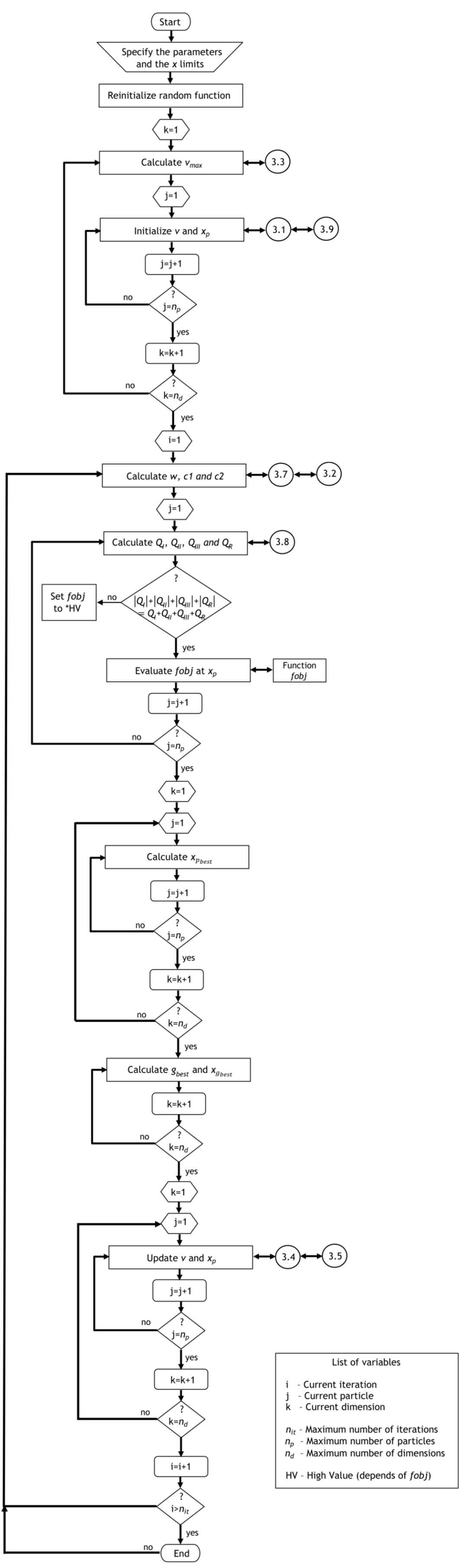

Figure 4.1 Block diagram of the PSO algorithm code.

5 TMB optimization

A TMB column with a section length of 2.1 dm and a diameter of 0.26 dm was considered. The feed concentration was 2.9 g/L of each enantiomer, the mass transfer coefficient, k_L, was 6 min^{-1}, the porosity, ε, was 0.4 and the Peclet number, *Pe*, was 2000. The operation was performed at 303.15 K.

In the remaining of this chapter, ten runs will be made for each optimization and the penalty coefficient, ω, will be changed in order to have the desired purities.

5.1 Optimization through productivity

First, the separation of the bi-naphthol enantiomers is going to be optimized with the objective of maximizing the productivity ($g/L_{adsorbent}/day$). The optimization variables are the eluent, Q_E, the extract, Q_X , the recycle, Q_{IV} , the feed, Q_{feed} and the solid, Q_s, volumetric flow-rates. Each particle will then represent a set of five flow-rates. The number of dimensions and the degrees of freedom is then equal to five.

The raffinate and extract productivities are respectively given by

$$Prod_R = \frac{Rec_R Q_{feed} C_A^{feed}}{(1-\varepsilon) V_c N_c} \tag{5.1}$$

$$Prod_X = \frac{Rec_X Q_{feed} C_B^{feed}}{(1-\varepsilon) V_c N_c} \tag{5.2}$$

where Rec_X and Rec_R are the extract and the raffinate recoveries, respectively, C_A^{feed} and C_B^{feed} are the feed concentration (massic) of A and B, respectively, ε is the bulk porosity, V_c is the column volume and N_c is the number of columns (in the TMB case, N_c=1) .

The raffinate and extract recoveries are respectively determined by

$$Rec_R = \frac{Q_R C_A^R}{Q_{feed} C_A^{feed}} \tag{5.3}$$

$$Rec_X = \frac{Q_x C_B^X}{Q_{feed} C_B^{feed}} \tag{5.4}$$

where C_A^R and C_B^X are the massic concentrations of A and B in the raffinate and extract streams, respectively.

Here, the productivity that is considered in the objective function is expressed by

$$Prod = Prod_X + Prod_R \tag{5.5}$$

According with the final use of the bi-naphtol enantiomers once they are separated, specified

purities (P^{set}) are required. The raffinate and extract purities are then used as constraints and can respectively be determined by

$$P_R = \frac{C_A^R}{C_B^R + C_A^R} \tag{5.6}$$

$$P_X = \frac{C_B^X}{C_B^X + C_A^X} \tag{5.7}$$

Finally, the objective function can be expressed by

$$fobj = \frac{1}{1+Prod} + \omega \sum_{i=1}^{2} f_i^2 \tag{5.8}$$

where ω is the penalty coefficient and f_i is given by

$$f_i = P_i - P^{set} - |P_i - P^{set}| \tag{5.9}$$

As it is intended to maximize the productivity, the objective function in Equation 5.8 will be minimized in the optimization process.

5.1.1 Results and Discussion

Wu et al. used the PSO-TVIW to optimize the TMB, using Table 5.1 parameters (2006). This optimization was repeated and the results were compared with ones obtained in their work.

Table 5.1 Optimization 5.1 parameters (flow-rates in mL/min).

	min	*max*
Q_E	10	30
Q_X	10	30
Q_{IV}	10	40
Q_{feed}	3	5
Q_s	8	14
n_{it}	50	
n_d	5	
n_p	50	
c1	0.5	
c2	0.5	
ω	105	
w_0	1.2	
w_f	0.1	
P^{set}	0.97	
factor	2	

The results obtained in ten runs are compared with Wu et al. (2006) in Table 5.2 (although the number of runs used by those authors is not referred).

Table 5.2 Comparison between *Optimization 5.1* and Wu et al. (2006),(flow-rates in mL/min, productivity in g/L_{ads}/day, CPU time in hours).

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	Prod	$fobj \times 10^2$	CPU
Wu *et al.*	22.1	19.6	21.5	4.4	9.0	99.3	97.6	144.0	94.3	-
Run1	29.8	25.7	23.4	**	10.4	98.9	98.6	155.1	90.3	1.4
Run2	**	28.6	21.3	**	9.5	99.1	98.3	155.4	90.3	1.5
Run3	29.5	18.8	13.9	**	9.5	98.9	98.6	154.2	90.3	1.5
Run4	20.5	15.8	18.5	**	8.9	98.8	98.5	154.4	90.3	1.3
Run5	**	25.5	23.6	**	10.5	98.7	98.7	154.9	90.3	1.3
Run6	29.9	28.5	21.3	**	8.9	99.4	97.9	155.3	90.3	1.5
Run7	**	28.7	20.5	**	8.5	99.2	98.1	155.3	90.3	1.5
Run8	27.1	25.7	21.3	**	8.8	98.8	98.5	155.3	90.3	1.1
Run9	29.9	26.0	25.8	**	11.0	98.7	98.6	155.0	90.3	0.9
Run10	29.4	24.7	16.3	**	8.3	99.1	98.2	154.5	90.3	1.5

** maximum was attained (200 mL/min)

It is possible to verify that the results in this study are better than the published ones (the productivity increased 7%) and the purities constraints were respected. This migth be explained by the algorithm changes in the calculation of x_p^{i+1} (see section 3.1) or due to numerical aspects, an analysis is given in Appendix D.

These study results show that different sets of variables correspond to the same value of objective function. The fact that the initial family of particles is random and that there is a random term in the PSO algorithm explains the results difference.

Table 5.2 shows that the particles are hitting the limits, which means that the exploitation zone is not big enough and so other optimal points might exist (the objective function might have multiple local minimums). To study the eventual existence of other minimums, the maximum and minimum limits of each dimension were enlarged (max, 200 mL/min and min, 0.01 mL/min). All the other parameters were maintained except the number of iterations which

was set to 500 (test runs showed that to a bigger exploitation zone a minimum number of 500 iterations was needed). The results are presented in Table 5.3.

Table 5.3 Optimization 5.2 results (flow-rates in mL/min, productivity in $g/L_{ads}/day$ and CPU time in hours). The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj \times 10^2$	CPU
Run1	191.0	148.3	1.4	5.9	15.9	97.0	97.0	178.5	89.0	4.5
Run2	**	169.1	*	6.8	12.0	97.0	97.0	206.4	87.5	4.0
Run3	123.4	86.2	*	9.3	14.1	86.0	97.3	263.9	597.5	3.6
Run4	199.5	158.2	*	6.2	15.1	97.0	97.0	189.1	84.4	5.1
Run5	169.5	135.0	*	6.8	13.1	97.0	97.0	205.2	87.5	4.0
Run6	**	136.4	*	18.6	26.6	89.1	84.7	503.4	976.3	4.3
Run7	**	169.6	*	6.8	11.9	97.0	97.0	206.3	87.5	5.0
Run8	199.7	195.4	27.3	6.9	12.2	97.0	97.0	211.4	87.2	3.5
Run9	**	171.5	*	6.7	11.2	97.0	97.0	204.2	87.6	5.5
Run10	**	157.2	*	6.1	15.5	97.0	97.0	185.4	88.6	4.4

* minimum was attained (0.01 mL/min)
** maximum was attained (200 mL/min)

Comparing these results with the ones in Table 5.2 it is visible that indeed there exist other local minimums since different results were obtained; in fact, the productivity increased 25%. However, the flow-rates present a large dispersion, some are still hitting the limits and in some runs the purities were not respected (that is why, in those cases, the *fobj* value is higher); this suggests that the PSO parameters (*c1*, *c2* and *w*) might need to be tuned. In fact those parameters must be adapted to each case and since the limits were enlarged the previous parameters might not be suited for a big exploitation zone. This will be studied in the next subchapter. Moreover, the algorithm is dependent on the initial system which is characterized by the randomness of the particles position; this means that the algorithm will find different minimums in every run. Depending on the objective function there might or not exist a tendency for the system to converge to a certain minimum.

The CPU increased as expected because the number of iterations was increased.

5.1.2 *c1*, *c2* and *w* influence

In *Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients*, Ratnaweera et al. (2004) say that starting with a large *c1* and a small *c2* allows the particles to better explore the space instead of going straight to the system's best (g_{best}). On the other side, the system should end with a small *c1* and a large *c2* to ensure the convergence.

In the remaining of this work, the values of *c1* and *c2* are respectively calculated at each iteration by

$$c1=\frac{(0.5-2.5)i}{n_{it}}+2.5 \tag{5.10a}$$

$$c2=\frac{(2.5-0.5)i}{n_{it}}+0.5 \tag{5.10b}$$

Here, *c1* was decreased from 2.5 to 0.5 and *c2* was increased from 0.5 to 2.5.

In *Inertia Weight Strategies in Particle Swarm Optimization* (Bansal et al., 2011) several equations to calculate *w* are summarized. In this thesis, the behavior of those equations was studied (i.e., the *w* value was plotted in function of the number of iterations in Figures 5.1 and 5.2) and compared with the original formula presented by Shi and Eberhart (1998),

$$w=w_0+(w_f-w_0)\frac{i}{n_{it}} \tag{5.11}$$

Moreover, Shi and Eberhart (2001) proposed a new version, the PSO-RANDIW (Random Inertia Weight), in which *w* should be calculated by

$$w=0.5+\frac{rand(1)}{2} \tag{5.12}$$

Here, *w* does not depend on the number of iterations.

Equation 5.12 presentes a total dispersion between 0 and 1, as shown in Figure 5.1.

In "Particles Swarm Optimization" (2006) Clerc suggests that *w* should be constant,

$$w=0.7 \tag{5.13}$$

In "Chaotic Inertia Weight in Particles Swarm Optimization", Feng et al. (2007) talk about a

$$w=(w_f-w_0)\frac{n_{it}-1}{n_{it}}+z_i w_f \tag{5.14}$$

$$z_i = 4z_{i-1}(1-z_{i-1}),\ z_0 = 0.35$$

new way to update *w* at each iteration by

Which has the same trend behavior than the equation proposed by Shi and Eberhart (1998), but with a larger dispersion (see Figure 5.1).

Moreover, they presented another equation which does not have a given initial and final value

$$w=0.5rand(1)+0.5z_i \tag{5.15}$$

$$z_i = 4z_{i-1}(1-z_{i-1}),\ z_0 = 0.35$$

and contains a random term,

Figure 5.1 shows that this equation has no tendency and presents a very large dispersion.

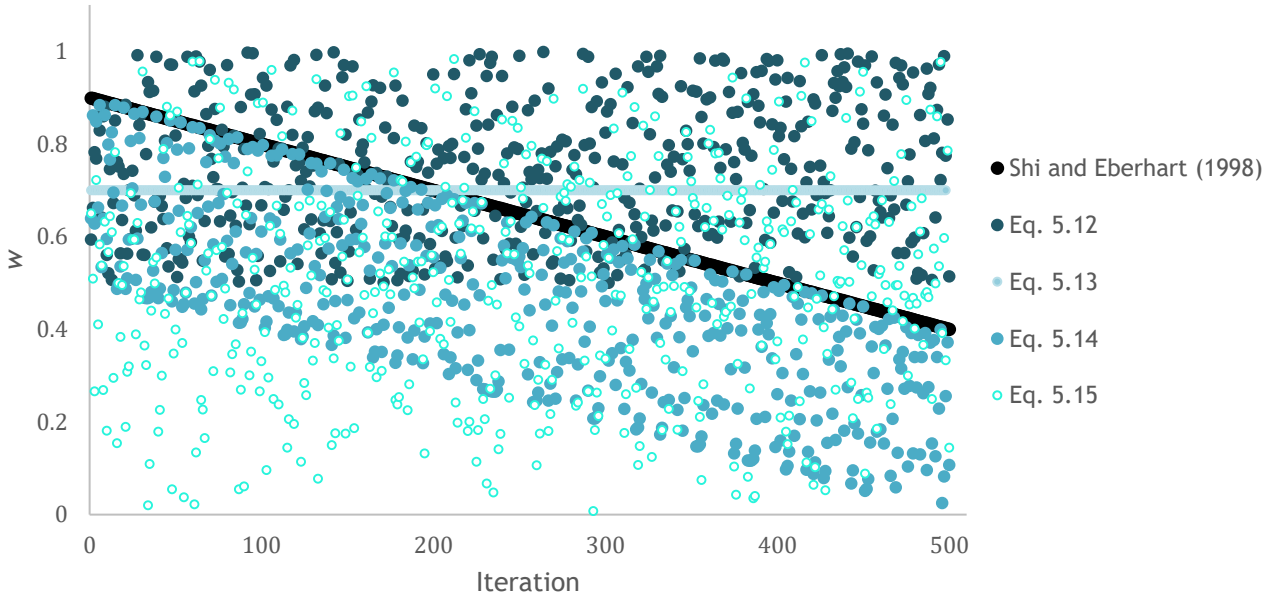

Figure 5.1 Comparison of the calculation of *w*.

Kentzoglanakis and Poole introduced the concept of the oscillating Inertia Weight, considering that "the swarm periodically transitions from exploratory to exploitatory states of search"

$$w=\frac{w_f+w_0}{2}+\frac{w_f-w_0}{2}\cos\frac{2\pi i}{T} \tag{5.16}$$

$$T=\frac{2\left(\frac{3n_{it}}{4}\right)}{3+2\times7}$$

(Kentzoglanakis and Poole, 2009); *w* is then calculated by

$$w=w_f+(w_0-w_f)0.95^{i-1} \tag{5.17}$$

Hassan et al. (2006) proposed that *w* should decrease, according to the equation

$$w=w_f+(w_0-w_f)e^{-\left(\frac{i}{10n_{it}}\right)^2} \tag{5.18}$$

A similar behavior was developed by Chen et al. (2006) who wrote that *w* could be given by

Figure 5.2 shows that these two equations have the same initial and final inertia weight value when compared with Shi and Eberhart (1998) but the decreasing behavior is very different.

The influence that each particle's best ($x_{p_{best}}$) exerts on the optimization process, using the PSO method is called by some authors as the "local search"; on the other hand, the effect of the overall best particle ($x_{g_{best}}$) is considered as the "global search". To balance the local and

$$w = (w_0 - w_f - d1)e^{\frac{1}{1+\frac{id2}{n_{it}}}} \tag{5.19}$$

$$d1=0.2\ ,\quad d2=0.75$$

the global searches, Li (2009) proposed that w should be given by

Unlike the previous equations, Gao et al. (2008) suggest that w should increase as the

$$w = w_f + (w_0 - w_f)\log_{10}\left(a + 10\frac{i}{n_{it}}\right) \tag{5.20}$$

$$a=5$$

optimization attains its end. In this case, w is calculated by

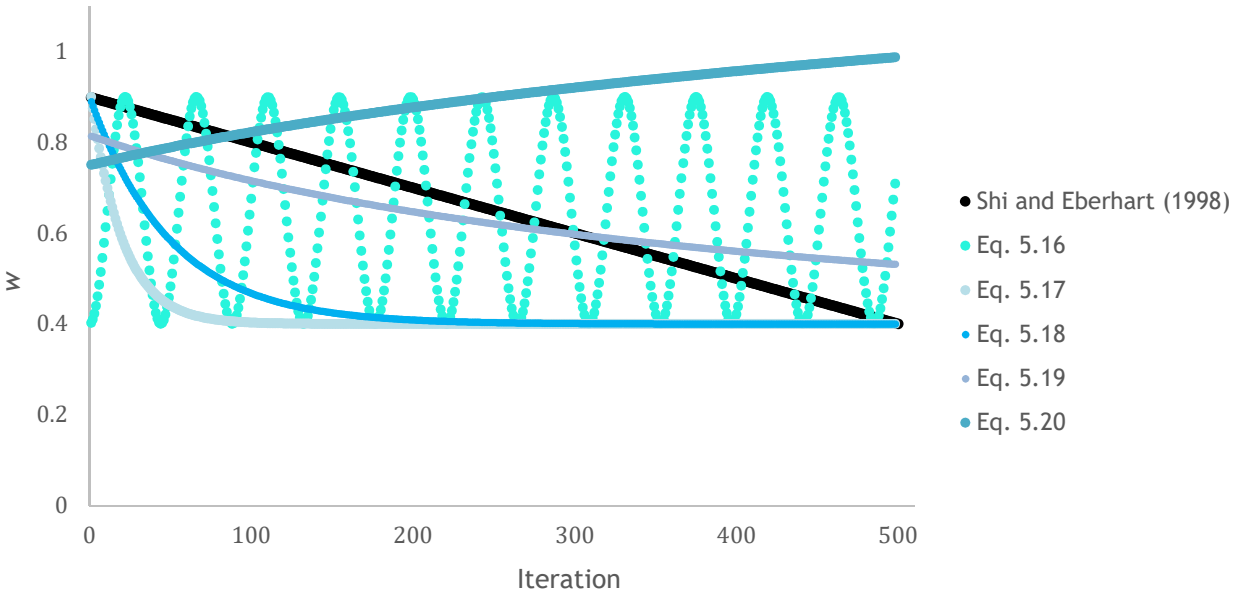

Figure 5.2 Comparison of the calculation of w.

Equations 5.11, 5.12 and 5.14 were tested, and the best results were obtained, using Shi and Eberhart's equation (1998) (this analysis is given in Appendix C).

Concerning the parameters *w0* and *wf*, previous works show that better results can be obtained, setting the value of the initial and final inertia weights to 0.9 and 0.4, respectively (Ratnaweera

et al., 2004; Eberhart and Shi, 2001; Shi and Eberhart, 1998). In the remaining of this work, those values will then be used as well as Equation 5.11 for the *w* calculation.

As the parameters *c1* and *c2* will no longer be constant, the algorithm is the PSO-TVAC (previously, the PSO-TVIW was used). *Optimization 5.2* was repeated with those changes and the results are presented in Table 5.4.

Table 5.4 Optimization 5.3 results. (flow-rates in mL/min, productivity in g/L_{ads}/day and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj \times 10^2$	CPU
Run1	167.1	137.7	1.4	6.8	12.0	97.0	97.0	206.6	87.5	4.0
Run2	**	167.7	*	6.8	12.5	97.0	97.0	206.9	87.4	3.8
Run3	**	167.0	*	6.8	12.7	97.0	97.0	206.1	87.5	4.1
Run4	**	178.1	12.1	6.8	13.0	97.0	97.0	205.6	87.5	4.3
Run5	**	170.0	*	6.7	11.8	97.0	97.0	203.7	87.6	3.5
Run6	**	169.4	*	6.9	12.0	97.0	97.0	207.8	87.4	3.3
Run7	**	163.9	*	6.7	13.6	97.0	97.0	203.4	87.6	4.5
Run8	198.9	165.6	*	6.8	12.9	97.0	97.0	206.6	87.5	4.0
Run9	129.5	100.6	4.1	6.4	12.5	97.0	97.5	193.7	88.1	3.9
Run10	189.8	179.4	22.9	7.0	12.7	97.0	97.2	203.8	87.6	3.5

* minimum was attained (0.01 mL/min)
** maximum was attained (200 mL/min)

As expected, the change in the calculation of *w*, *c1* and *c2* had a positive effect. These parameters will then be calculate in the same manner in the remaining of this work.

Unlike Table 5.3 results which had a large dispersion, Table 5.4 shows that the values of the feed and solid flow-rates are similar from run to run, this means that the objective function has a tendency to find a local minimum. Unlike Table 5.2 results, here the purities are respected in every run.

It is possible to verify that when the value of the eluent flow-rate reaches its maximum, the recycle flow-rate (i.e., TMB's section IV flow-rate) reaches its minimum. In fact, a large amount of fresh eluent to regenerate the solid is only required if a small amount of eluent is being recycled from section IV to section I, according to the following equation (see subchapter 3.2)

$$Q_I = Q_E + Q_{IV} \tag{5.21}$$

Table 5.3 also shows a tendency of the extract flow-rate to increase with the eluent flow-rate, this happens in order to achieve the purities constraints.

As the eluent and recycle flow-rates are mostly hitting the limits, a productivity based objective function does not allow to optimize the TMB's section I flow-rates (i.e., eluent and recycle).

For this purpose, the minimization of the eluent consumption will be introduced in the objective function further on this work.

5.2 Optimization through Eluent consumption

As a chiral separation is being performed, the eluent is also the mobile phase which means it is present in the feed, the eluent consumption, *EC*, is then calculated by

$$EC = \frac{Q_E + Q_{feed}}{Q_{feed} \sum_{i=1}^{nc} c^{feed}} \tag{5.22}$$

in which *nc* is the number of components (Rodrigues et al., 2015; Nogueira et al., 2016).

In order to limit the eluent consumption, the objective function can then be expressed by

$$fobj = EC + \omega \sum_{i=1}^{2} f_i^2 \tag{5.23}$$

As previously seen, the maximum of productivity is obtained around 206 $g/L_{ads}/day$ to which correspond values of 6.8 mL/min and 12.6 mL/min for the feed and solid flow-rates, respectively. In order to minimize the eluent consumption, the maximum and minimum values for Q_{feed} and for Q_s were set between 6.3 and 7.3 mL/min and between 12.1 and 13.1 mL/min, respectively. The limits for the other flow-rates were the ones used in the previous subchapter.

5.2.1 Results and Discussion

To perform the current Optimizations, the parameters are listed in Table 5.5. *c1* and *c2* are calculated with Equations 5.10a and 5.10b, respectively. The results are presented in Table 5.6.

Table 5.5 Optimization 5.4 parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	6.3	7.3
Q_s	12.1	13.1
n_{it}	500	
n_d	5	
n_p	50	
ω	1000	
w_0	0.9	
w_f	0.4	
P^{set}	0.97	
factor	2	

Table 5.6 Optimization 5.4 results. (flow-rates in mL/min, productivity in g/L_{ads}/day, eluent consumption in dL/g and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	*fobj*×10^2	*EC*	CPU
Run1	30.9	23.1	23.3	7.0	12.2	97.0	97.0	210.8	96.0	93.8	1.7
Run2	23.1	20.3	29.3	6.5	12.4	97.0	97.0	198.9	78.6	78.2	1.8
Run3	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.7
Run4	22.4	19.8	29.6	6.3	12.3	97.0	97.0	193.0	78.2	78.0	1.7
Run5	21.4	18.7	28.6	6.4	12.1	97.0	97.0	194.3	75.5	75.0	1.7
Run6	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.7
Run7	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.7
Run8	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.8
Run9	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.8
Run10	21.0	18.3	28.8	6.3	12.1	97.0	97.0	191.3	75.3	74.8	1.8

As expected, adding the eluent consumption and reducing the search region for the feed and the solid flow-rates, all the flow-rates were optimized. In fact, the objective function in Equation 5.23 is directly related with the eluent consumption which prevents the system from increasing the eluent flow-rate until its maximum. Moreover, the feed and the solid flow-rates still had a (although small) search region which allows their adaptation to a smaller eluent consumption; like this, the productivity is not significantly affected (only 6.8% reduction against an 85% reduction in the eluent consumption).

Once more, the purities constraints were always respected.

Here, the CPU time was smaller. Although the number of iterations was the same, the feed and the solid flow-rates had a small searching region that was already defined which means that all the points for dimensions four and five are possible, like this, the total number of non-possible points was reduced (in the case of non-possible points, the model takes longer to run).

Optimizations 5.3 and 5.4 can be considered as a *Two-steps optimization* since the feed and the solid flow-rates were first optimized (*Step1*) and then the eluent, the extract and the recycle flow-rates were optimized (*Step2*).

5.3 Optimization through Productivity and Eluent Consumption

In this subchapter a new strategy to optimize the system is presented. The objective was to build an objective function that allows to optimize the system's productivity and eluent consumption at the same time. As it is desired to maximize the productivity and to minimize the eluent consumption, both together may be expressed as the minimization of the objective function given by

$$fobj=EC + \frac{1}{0.01+Prod} + \omega \sum_{i=1}^{2} f_i^2 \qquad (5.24)$$

Subject to

$[Prod; EC] \geq 0$

$1 \geq [P_R; P_X] \geq 0$

The purities correction is made using penalties as already seen. In order to avoid numerical problems, a factor of 0.01 was added and some constraints were used.

5.3.1 Results and Discussion

As two parameters are being optimized at the same time, the number of iterations was increased to 2000.

As seen in chapter 3.1, *w* depends on the number of iterations, the larger is n_{it}, the slower is the convergence; since n_{it} was increased it was expected that the convergence would be slower, which was verified through some test runs. In order to optimize those results, *w* was calculated by Equation 5.11 until iteration number 500, after what it was kept constant. The same strategy was applied to the calculation of *c1* and *c2*.

The parameter ω was increased to 4000 to ensure that the purities constraints were respected. Through test runs it was verified that v_{max} was too big, which was not letting the particles explore correctly; the factor was then set to five (see Equation 3.3) . Table 5.7 summarizes the optimization parameters.

Table 5.7 Optimization 5.5 parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	0.01	200
Q_s	0.01	200
n_{it}	2000	
n_d	5	
n_p	50	
ω	4000	
w_0	0.9	
w_f	0.4	
P^{set}	0.97	
factor	5	

The next table shows the optimization results.

Table 5.8 Optimization 5.5 results (flow-rates in mL/min, productivity in g/L_{ads}/day, eluent consumption in dL/g and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj \times 10^2$	*EC*	CPU
Run1	28.8	25.3	23.3	7.0	12.2	97.0	97.0	213.5	726.2	87.7	8.2
Run2	27.9	24.4	27.8	7.0	12.2	97.0	97.0	212.8	725.7	85.9	8.1
Run3	27.3	24.0	27.5	7.0	12.0	97.0	97.0	212.2	725.4	84.6	7.9
Run4	29.6	26.0	28.4	7.1	12.5	97.0	97.0	214.5	727.5	89.4	8.2
Run5	28.2	24.7	27.7	7.0	12.2	97.0	97.0	213.7	726.0	86.2	8.0
Run6	51.5	23.6	1.1	6.8	11.4	97.0	97.0	206.5	805.1	147.1	7.5
Run7	27.4	24.1	27.6	7.0	12.1	97.0	97.0	212.3	725.5	84.9	8.2
Run8	27.6	24.2	27.5	7.0	12.1	97.0	97.0	212.4	725.5	85.1	8.0
Run9	27.8	24.4	27.6	7.0	12.1	97.0	97.0	213.1	725.7	85.5	8.0
Run10	27.9	24.4	27.9	7.0	12.2	97.0	97.0	212.7	725.7	85.8	8.0

Comparing the results in Table 5.8 with the ones in Table 5.7 it is possible to conclude that an objective function comprising the productivity and the eluent consumption is better than optimizing separately, since a higher productivity was obtained. The eluent consumption was also higher, but as the main objective of this work is to optimize the productivity (without having an excessive eluent consumption), this result is considered better than the previous one.

The solid and the recycle flow-rates have the same average values than the ones in Table 5.7. The feed and the eluent (and the extract because, as previously seen, it increases if the eluent flow-rate increases) flow-rates have slightly increased.

The CPU time is higher because the number of iterations increased.

The purities constraints were respected which indicates that the penalty coefficient was properly increased.

In this subchapter only one step was needed to optimize the system, this strategy can thus be called *Single optimization*.

5.4 *Parallel* PSO

Since its first appearance, variants of the original PSO were developed as referred by Ratnaweera et al. (2004). In the past few years, studies on multiobjective optimization increased and it has already been applied to the PSO method (Sha and Lin, 2010) and to the SMB and Varicol optimizations (Zhang, Hidajat and Ray, 2002).

In this work, a new way of optimizing, using the PSO method, is presented. As seen in the previous subchapters, the feed and the solid flow-rates can be optimized, using a productivity based objective function. On the other hand, the eluent, the extract and the recycle flow-rates are better optimized, using the eluent consumption. These conclusions can be easily explained, looking to the TMB model and to the calculation of the productivity and the eluent consumption, already mentioned. In this subchapter, a variant of the PSO method that uses two different objective functions in parallel was used. As previously seen, the TMB is optimized through the manipulation of the eluent, the extract, the recycle, the feed and the solid flow-rates and the number of dimensions is then five. Here, the first three dimensions will be optimized by the eluent consumption (Equation 5.25) and the last two by the productivity (Equation 5.26).

$$fobj_E = EC + \omega_E \sum_{i=1}^{2} f_i^2 \tag{5.25}$$

$$fobj_P = \frac{1}{1+Prod} + \omega_P \sum_{i=1}^{2} f_i^2 \tag{5.26}$$

Instead of one *step*, *v*, the *Parallel* PSO will have two different *steps*: the *eluent-step*, v_E, and the *productivity-step*, v_P. In the equations that will be used to update the *steps*, the particle's dimension must be included, as follows:

$$v_E^{i+1,1:3} = wv_E^{i,1:3} + c1rand(1)(x_{p_{best}}{}^{i} - x_p^{i,1:3}) + c2rand(1)(x_{g_{best\,E}}{}^{i,1:3} - x_p^{i,1:3}) \tag{5.27}$$

$$v_P^{i+1,4:5} = wv_P^{i,4:5} + c1rand(1)(x_{p_{best}}{}^{i} - x_p^{i,4:5}) + c2rand(1)(x_{g_{best\,P}}{}^{i,4:5} - x_p^{i,4:5}) \tag{5.28}$$

Also, there exist now two best particles, $x_{g_{best\,E}}{}^{i,1:3}$ and $x_{g_{best\,P}}{}^{i,4:5}$.

The update of x_p is done, using both v_E and v_P

$$x_p^{i+1,1:3} = x_p^{i,1:3} + v_E^{i+1,1:3} \tag{5.29}$$

$$x_p^{i+1,4:5} = x_p^{i,4:5} + v_P^{i+1,4:5} \tag{5.30}$$

As shown by Equations 5.27 and 5.28, the system will have two best points at each iteration, $x_{g_{best\,E}}$ and $x_{g_{best\,P}}$.

The algorithm is summarized in Figure 5.3.

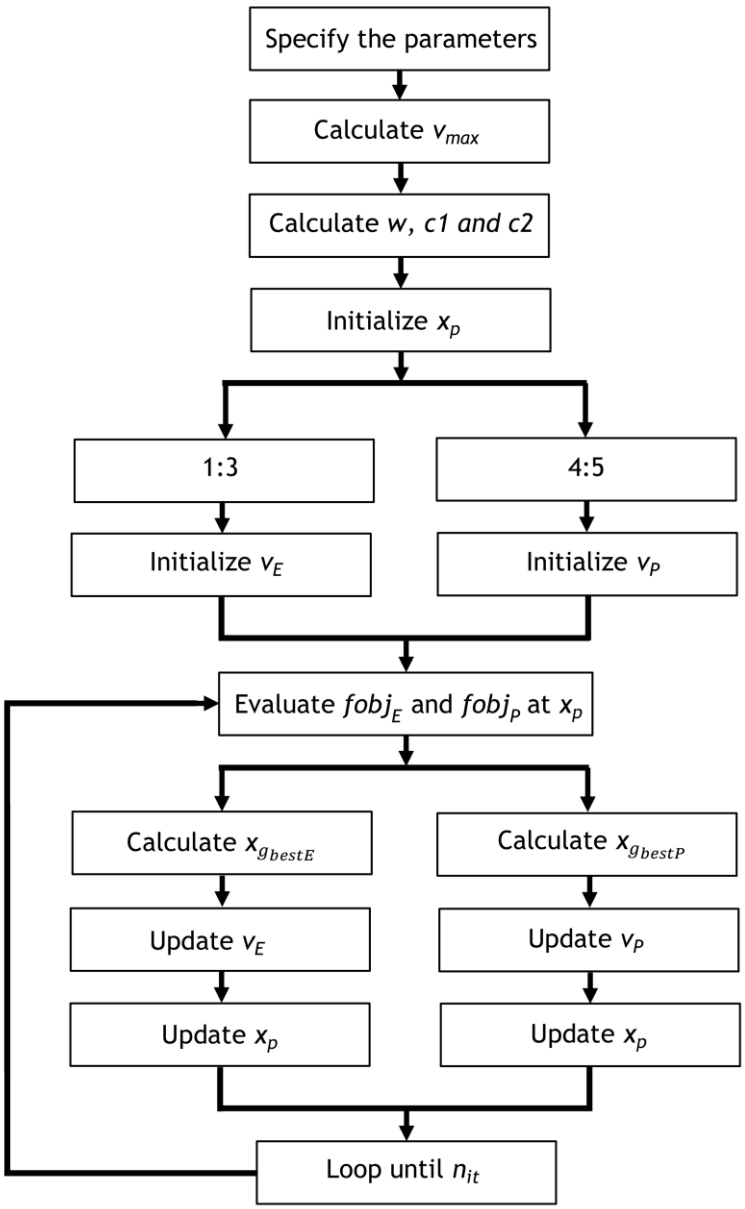

Figure 5.3 Parallel PSO scheme.

5.4.1 Results and Discussion

The optimization parameters are listed in Table 5.9 and the results are in the Tables 5.10 and 5.11.

Table 5.9 Optimization 5.5 parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	0.01	200
Q_s	0.01	200
n_{it}	1000	
n_d	5	
n_p	50	
ω_E	4000	
ω_P	5000	
w_0	0.9	
w_f	0.4	
P^{set}	0.97	
factor	5	

Table 5.10 Eluent consumption-based objective function *Optimization 5.6* results. (flow-rates in mL/min, productivity in g/L_{ads}day, eluent consumption in dL/g and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj_P \times 10^2$	$fobj_E \times 10^2$	*EC*	CPU
Run1	25.7	22.5	29.1	6.7	12.5	97.0	97.0	205.0	87.6	83.0	82.9	6.6
Run2	22.8	20.8	27.1	6.7	11.4	97.0	97.0	203.4	87.9	76.4	76.1	6.9
Run3	21.2	17.7	24.1	6.5	10.9	97.0	97.0	197.2	88.1	73.6	73.5	6.8
Run4	26.6	23.6	28.2	6.9	12.1	97.0	97.0	209.2	88.4	85.2	83.9	6.9
Run5	20.2	18.2	26.4	6.5	11.1	97.0	97.0	197.6	88.1	70.9	70.8	6.5
Run6	24.8	22.4	24.7	6.7	10.8	97.0	97.0	203.9	87.6	81.1	81.1	6.6
Run7	26.5	23.4	28.9	6.8	12.3	97.1	97.0	205.7	87.5	84.8	84.8	6.7
Run8	26.1	22.8	28.8	6.8	12.4	97.0	97.0	206.3	87.5	83.6	83.6	6.5
Run9	17.3	16.0	24.2	6.2	10.1	97.0	97.1	189.6	88.7	65.5	65.1	6.6
Run10	21.9	19.3	26.9	6.6	11.6	97.0	97.0	201.1	87.9	74.6	74.4	6.6

Table 5.11 Productivity-based objective function *Optimization 5.6* results. (flow-rates in mL/min, productivity in g/L_{ads}day, eluent consumption in dL/g and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj_P \times 10^2$	$fobj_E \times 10^2$	*EC*	CPU
Run1	29.0	25.3	28.9	6.9	12.5	97.0	97.0	208.9	87.3	89.9	89.9	6.6
Run2	39.2	31.6	27.0	6.7	13.1	97.1	97.2	204.3	87.6	117.9	117.9	6.9
Run3	26.2	23.1	26.6	6.8	11.6	97.0	97.0	207.6	87.4	83.3	83.3	6.8
Run4	36.7	31.5	28.3	6.9	12.9	97.0	97.1	209.5	87.3	109.2	109.2	6.9
Run5	22.0	19.9	26.4	6.6	11.2	97.2	97.0	194.3	87.9	74.8	74.8	6.5
Run6	70.4	62.5	26.0	6.8	13.0	97.0	97.0	208.2	87.4	194.7	194.7	6.6
Run7	33.5	29.5	28.6	6.9	12.6	97.0	97.0	210.6	87.2	100.7	100.7	6.8
Run8	29.1	25.3	28.6	6.8	12.5	97.0	97.1	208.0	87.4	90.5	90.5	6.5
Run9	39.1	28.1	24.6	6.7	13.5	97.2	97.0	203.7	87.5	74.6	117.8	6.6
Run10	24.9	21.7	26.8	6.8	11.7	97.0	97.0	206.6	87.5	80.3	80.3	6.6

In the case of the *Parallel* PSO the results are in duplicate because the system has an optimum point that was obtained by the eluent consumption-based objective function ($x_{g_{best\,E}}$ at the last iteration) and another one obtained by the productivity-based objective function ($x_{g_{best\,P}}$ at the last iteration). The results of the productivity-based objective function have slightly higher productivity and eluent consumption.

Comparing the *Parallel* PSO with the *Two-parts optimization* (subchapter 5.2.1) and with the *Single optimization* (subchapter 5.3.1), the results are similar which means that the *Parallel* PSO is another strategy to optimize this system.

In the previous strategies (*Two-Parts* and *Single optimization*) the algorithm started to optimize the feed and the solid flow-rates and just once they were almost in its optimum values, it would start to optimize the eluent, extract and recycle flow-rates. Here, as the division of the flow-rate's optimization was implemented in the algorithm (and was not just dependent on the objective function), all the flow-rates started to be optimized at the same time which allowed to halve the number of iterations; the CPU time was then decreased.

The purities constraints were respected.

5.5 Comparison and conclusions

In Chapter 5, the TMB unit was optimized, using several strategies. In subchapters 5.1 and 5.2 the optimization was done in two parts, first the feed and the solid flow-rates were optimized, using a productivity-based objective function; then the eluent, extract and recycle were optimized, using an eluent consumption-based objective function. In subchapter 5.3, an objective function comprising the productivity and the eluent consumption was used. Finally, in subchapter 5.4, a variant of the PSO method, the *Parallel* PSO, was used.

The results show that in each set of runs there is an outlier; this is explained by the randomness of the algorithm and also by its dependence on the initial particles system.

Since the results obtained by the mentioned strategies respect the purities constraints and as the main objective is to maximize the productivity, here the comparison will be done in terms of that variable. Another important parameter is the CPU time which will also be part of the comparison.

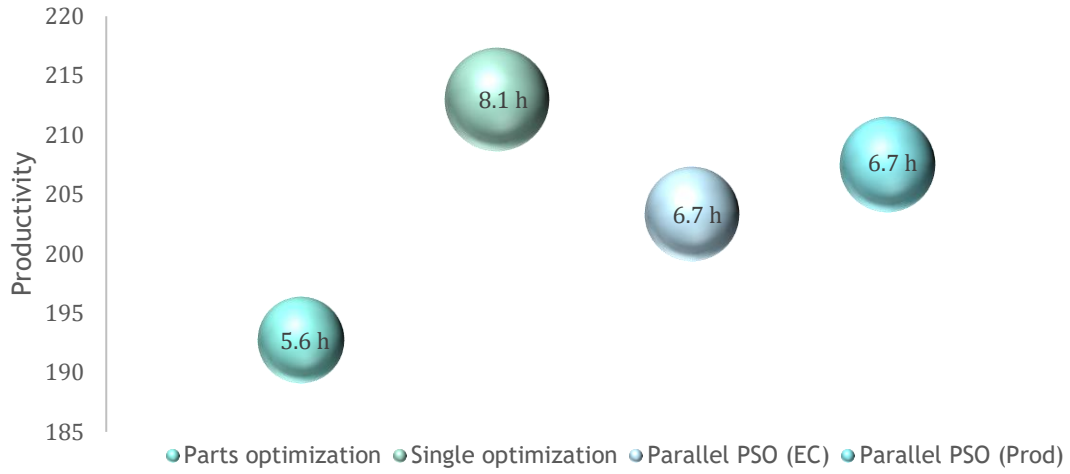

Figure 5.4 Optimization techniques comparison in terms of system's productivity *vs* CPU time (productivity in $g/L_{ads}/day$).

Although the *Parts optimization* is faster, it attains a lower productivity. The *Single optimization* presents the highest productivity, but also the larger CPU time. The *Parallel* PSO strategy conduces to productivities almost as high as the *Single optimization* ones, using a smaller CPU time; this strategy is then the best compromise between the productivity and the CPU time.

6 SMB simulation

In the remaining of this chapter, a SMB device with the total bed length of 8.4 dm and a column diameter of 0.26 dm is considered. The feed concentration is 2.9 g/L of each enantiomer, the mass transfer coefficient, k_L, is 6 min^{-1} , the porosity, ε, is 0.4 and the Peclet number, *Pe*, is 2000. The operation is performed at 303.15 K.

The flow-rates obtained in run7 of *Optimization 5.6* were used to simulate the SMB unit with four, eight and twelve columns (one, two and three columns per section, respectively).

The SMB flow-rates and the switching time were obtained, using the equations in subchapter 3.4.

The productivity, the raffinate and the extract purities were, respectively, calculated by

$$Prod = \frac{Q_R \int_t^{t+N_c t^*} c_A^R \, dt}{(1-\varepsilon) V_c N_c t^*} + \frac{Q_X \int_t^{t+N_c t^*} c_B^X \, dt}{(1-\varepsilon) V_c N_c t^*} \tag{6.1}$$

$$P_R = \frac{\int_t^{t+N_c t^*} c_A^R \, dt}{\int_t^{t+N_c t^*} c_A^R \, dt + \int_t^{t+N_c t^*} c_B^R \, dt} \tag{6.2}$$

$$P_X = \frac{\int_t^{t+N_c t^*} c_B^X \, dt}{\int_t^{t+N_c t^*} c_A^X \, dt + \int_t^{t+N_c t^*} c_B^X \, dt} \tag{6.3}$$

The simulation parameters are in Table 6.1 and the results in Table 6.2.

Table 6.1 Simulation 6.1 operating conditions to a four column SMB** (flow-rates in mL/min, switching time in min).

Q_E	Q_X	Q_{IV}	Q_{feed}	t^*
27.4	24.1	35.7	7.0	5.5

** to the eight and the twelve column SMB, the switching time was divided by two and three, respectively

Table 6.2 Simulation 6.1 results (productivity in g/L_{ads}day, eluent consumption in dL/g).

	P_R	P_X	*Prod*	*EC*
SMB4	92.1	89.1	179.1	84.7
SMB8	96.4	94.8	199.7	84.7
SMB12	97.2	96.6	202.5	84.7
TMB	97.0	97.0	212.3	84.9

As expected, the results improve as the number of columns increases. Nevertheless, even with twelve columns, the purities constraints are not verified. A similar conclusion was obtained by Pais, Loureiro and Rodrigues (1998) who studied this system.

In the next figures, the internal concentration profiles for the bi-naphthol enantiomers separation, using the TMB or the SMB with four, eight or twelve columns is represented.

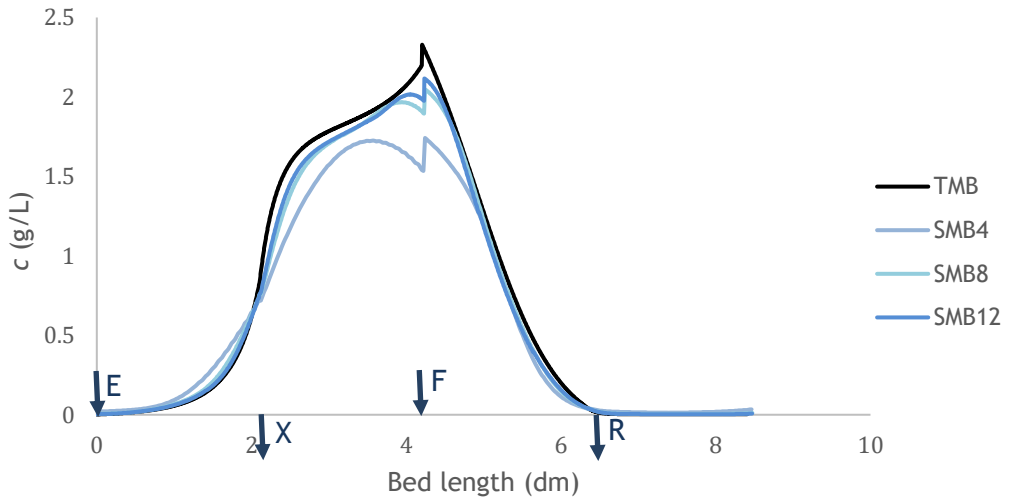

Figure 6.1 Cyclic steady-state internal concentration profiles of the more retained species for TMB and SMB at half of the switching time (E, eluent in; X, extract out; F, feed in; R, raffinate out).

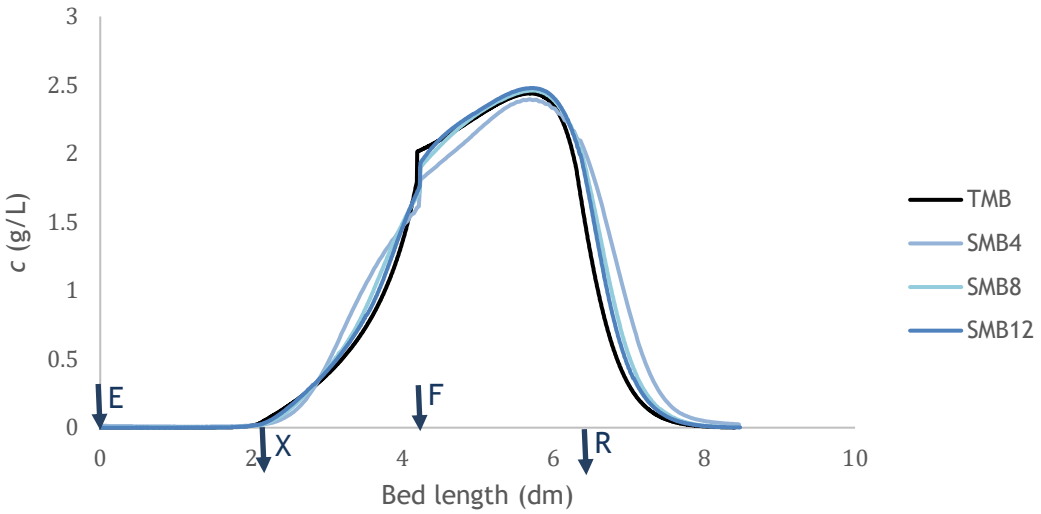

Figure 6.2 Cyclic steady-state internal concentration profiles of the less retained species for TMB and SMB at half of the switching time (E, eluent in; X, extract out; F, feed in; R, raffinate out).

Both Figures 6.1 and 6.2 show that the TMB profile has a more abrupt curve which avoids the concentration front from reaching the other component outlet; on the other hand, in the SMB the curve is smoother, consequently the outlet stream (raffinate ou extract) will be contaminated and the purities are lower.

7 TMB configuration optimization

In this chapter, the TMB configuration, i.e., the sections length, will be part of the optimization. For that purpose, the TMB equations were solved with dimensionless *z*, considering that the total length of the column, *L*, was given by

$$L=L_I+L_{II}+L_{III}+L_{IV} \tag{7.1}$$

in which *L* is equal to 8.4 dm.

Each particle will then represent the five flow-rates plus the lengths of sections I, II and III. The degrees of freedom increased to eight and the objective function is the one used in the *Single optimization* and is given by

$$\begin{aligned} &fobj=EC+\frac{1}{0.01+Prod}+\omega\sum_{i=1}^{2}f_i^2 \\ &\text{Subject to} \\ &[Prod;\ EC]\geq 0 \\ &1\geq[Pr;\ Px]\geq 0 \end{aligned} \tag{7.2}$$

The optimization parameters are listed in Table 7.1 and the results in Table 7.2.

Table 7.1 Optimization 7.1 parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	0.01	200
Q_s	0.01	200
L_I	0.01*L*	0.36*L*
L_{II}	0.01*L*	0.36*L*
L_{III}	0.01*L*	0.36*L*
n_{it}	2000	
n_d	8	
n_p	50	
ω	4000	
w_0	0.9	
w_f	0.4	
P^{set}	0.97	
factor	5	

Table 7.2 Optimization 7.1 results (flow-rates in mL/min, productivity in g/L_{ads}/day, eluent consumption is in dL/g, section's length in dm and CPU time in hours). The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	L_I	L_{II}	L_{III}	P_R	P_X	*Prod*	$fobj \times 10^2$	*EC*	CPU
Run1	80.8	49.5	13.6	9.9	17.5	1.8	3.0	3.0	97.0	97.0	299.5	560	93.5	19.8
Run2	47.5	36.9	24.8	7.9	14.0	1.7	2.0	2.0	97.0	97.0	238.6	647.7	72.5	20.3
Run3	75.4	51.0	21.1	9.8	17.6	1.6	3.0	3.0	97.0	97.0	294.9	559.1	89.1	18.3
Run4	76.9	57.9	22.5	9.7	16.4	1.2	3.0	3.0	97.0	97.0	294.2	564.0	83.9	20.4
Run5	77.9	48.0	13.2	9.9	16.9	1.8	3.0	3.0	97.0	97.0	300.0	558.8	90.4	20.1
Run6	74.5	50.5	20.9	9.8	17.4	1.6	3.0	3.0	97.0	97.0	295.2	557.6	88.1	18.3
Run7	78.9	49.5	13.9	9.9	17.0	1.9	3.0	3.0	97.0	97.0	297.8	556.6	92.0	18.2
Run8	76.6	49.9	19.4	9.8	17.8	1.8	3.0	3.0	97.0	97.0	295.5	559.3	90.2	19.0
Run9	83.1	46.1	8.5	9.9	17.7	2.1	3.0	3.0	97.0	97.0	299.2	560.8	95.9	18.4
Run10	81.2	52.8	15.9	9.9	17.3	1.8	3.0	3.0	97.0	97.0	299.4	87.9	93.9	18.2

As the columns sections length were free to vary, the system has more degrees of freedom which led to a better optimum. As shown in Table 7.2, the productivity increased significantly without compromising the eluent consumption. This proves that the TMB configuration is a determining factor for the separation.

Looking to the sections length, it is clear that section's IV length (L-L_I-L_{II}-L_{III}=L_{IV}) tends to be shorter than the others (0.6 dm against an average of 1.8 dm for section I and 3.0 dm for II and II); this means that, when considering the SMB, a small number of columns is needed to regenerate the eluent. Also, the number of columns of the separation zone (sections II and III) should be bigger.

Note that section's II and III lengths attained the maximum limit. In this case, it is acceptable because the limits were imposed in order to represent physically the system and it would not make sense to have a section with more than 36% of the total length.

7.1 SMB simulation

As seen in the previous subchapter, the average configuration obtained was 1.8|3|3|0.6; normalizing in order to obtain an integer number of columns per section, the configuration is 3|5|5|1. A SMB simulation was performed, considering this configuration (total number of

columns equal to fourteen), Run1 flow-rates and the parameters mentioned in chapter 6. The simulation parameters are listed in Table 7.3 and the results are in Table 7.4.

Table 7.3 Simulation 7.1 operating conditions (flow-rates in mL/min, switching time in min).

Q_E	Q_X	Q_{IV}	Q_{feed}	t^*
80.8	49.5	25.3	9.9	1.1

Table 7.4 Simulation 7.2 results and comparison with TMB (productivity in g/L_{ads}day, eluent consumption in dL/g).

	P_R	P_X	*Prod*	*EC*
SMB14	97.0	96.5	288.0	93.4
TMB (Run1)	97.0	97.0	299.5	93.5

Comparing these results with the SMB12 in Chapter 6 (the SMB4 and SMB8 are not comparable because the number of columns is very different), it is noticeable that in the SMB14, using a different number of columns per section, not only the purities constraints were almost respected, but also the productivity was significantly higher. The internal concentration profiles are represented in the next figure.

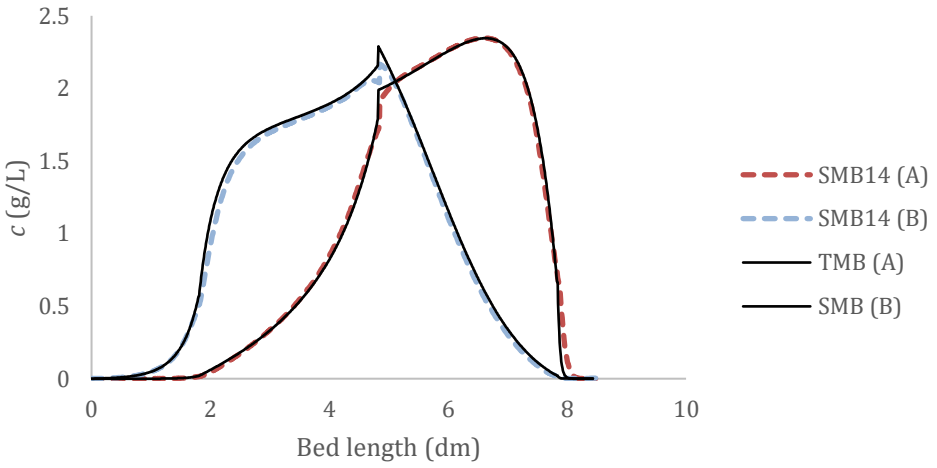

Figure 7.1 Cyclic steady-state internal concentration profiles of the less (A) and the more (B) retained species for TMB and SMB14 at half of the switching time.

As expected, the profiles are almost overlapped; in fact, the purities constraints were respected for the raffinate stream and were not very far for the extract stream. Unlike the profiles shown in Chapter 6, here the SMB curve is not significantly smoother which proves the lack of contamination.

7.2 Varicol simulation

As already mentioned in this work, the Varicol is a variant of the SMB with asynchronous shift of the inlet/outlet streams, i.e., the switching time is not the same for all the streams.

In the previous subchapter, fourteen columns were needed to respect the configuration that was obtained, when optimizing the TMB with variable sections length. Here, the Varicol will be used to reduce the number of columns (but maintaining the total bed length); a seven column Varicol was, then, considered (the operating conditions are the same as subchapter 7.1). As seen in the previous subchapter, the configuration was 3|5|5|1; as it is intented to halve the number of columns, the configuration will be 1.5|2.5|2.5|0.5 wich means that the total switching time will have to be fractioned in order to guarantee the mentioned configuration. To that purpose, the switching time was fractioned into two; half the time the configuration was 1|3|2|1 and the other half it was 2|2|3|0. After the first half time, only the extract and the raffinate streams are switched; at the end of the second half, the eluent and feed streams are switched and the initial configuration is reset. Figure 7.2 shows schematically what happens to the sections length.

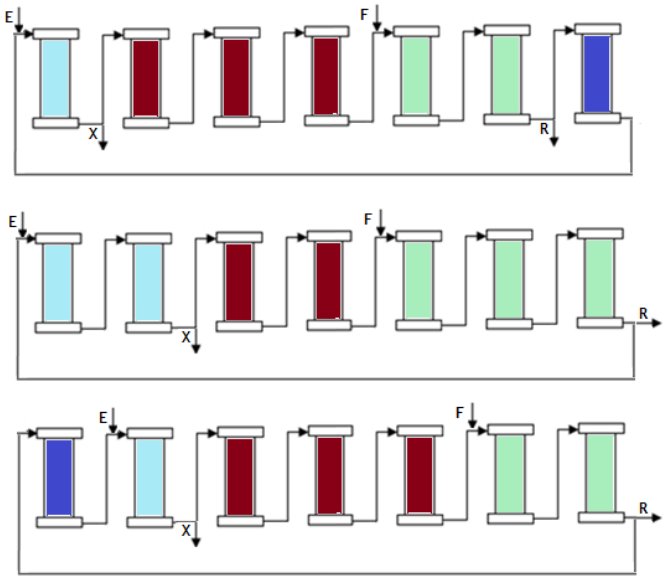

Figure 7.2 Varicol configuration scheme (section I in light blue, section II in red, section III in green and section IV in dark blue).

In Figure 7.2 it is visible that after the first switch, section IV disappears; in this case, the raffinate stream is collected before the dilution with the eluent stream (Ludemann-Hombourger et al., 2000).

The boundary condition and the node mass balance for section I are, respectively, given by

$$c_{iIII,L}\left(1-\frac{u_R}{u_{III}}\right)=\frac{u_I}{u_{III}}c_{iI,0} \tag{7.3}$$

$$u_E + u_{III} - u_R = u_I \tag{7.4}$$

This simulation results and the comparison with the SMB14 are in Table 7.5.

Table 7.5 Simulation 7.3 results and comparison with SMB14 (productivity in g/L_{ads}day, eluent consumption in dL/g).

	P_R	P_X	*Prod*	*EC*
Varicol	96.0	95.5	281.2	93.4
SMB14	97.0	96.5	288.0	93.4

Table 7.5 shows that the productivity is almost as high as in the SMB14. However, the purities are not respected yet, which might be explained by the much lower number of columns (as shown in Chapter 6, the SMB12 is the one that better approaches the TMB, a similar behavior might be expected in the Varicol since it is a variant of the SMB). The internal concentration profiles are represented in Figure 7.3.

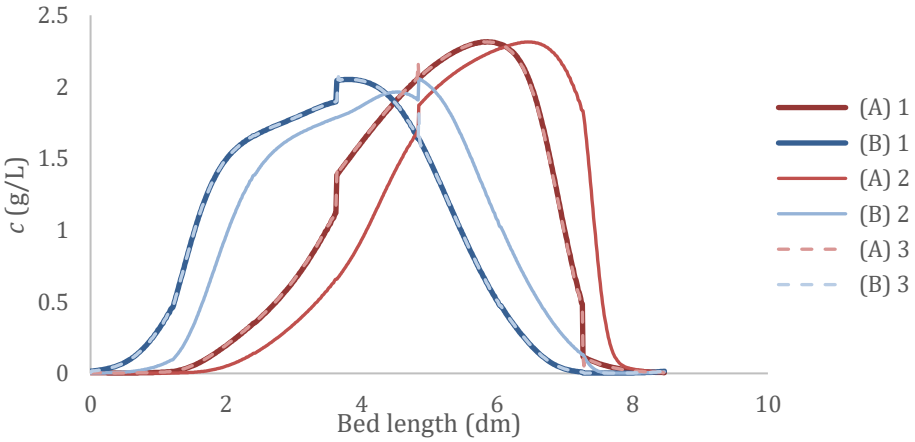

Figure 7.3 Steady-state internal concentration profiles of the less (A) and the more (B) retained species for Varicol at the switching time (1 and 3); half of the switching time (2).

The fact that there is a configuration change is visible in Figure 7.3. At the switching time the internal concentration profile is represented in (1); at half of the switching time the configuration changes and the profile changes too (2); after the next half the initial configuration is reset and so is the profile (3).

8 Objective function and algorithm analysis

This chapter is focused on the mathematical aspects of the presented objective functions and algorithm. For instance, Run9 of *Optimization 7.1,* Run6 of *Optimization 5.7* and Run4 of *Optimization 5.4* will be used. The next figures show all the points (i.e., all the sets of parameters for each particle at each iteration) in blue and the optimum point in red.

The objective functions are in the form

$$fobj = expression + penalty \tag{8.1}$$

in which the penalty is of the type

$$Penalty = \omega \sum_{i=1}^{2} f_i^2 \tag{8.2}$$

Like this, it was expected that plotting sets of parameters would give a parabolic shape as it is shown in the following figures.

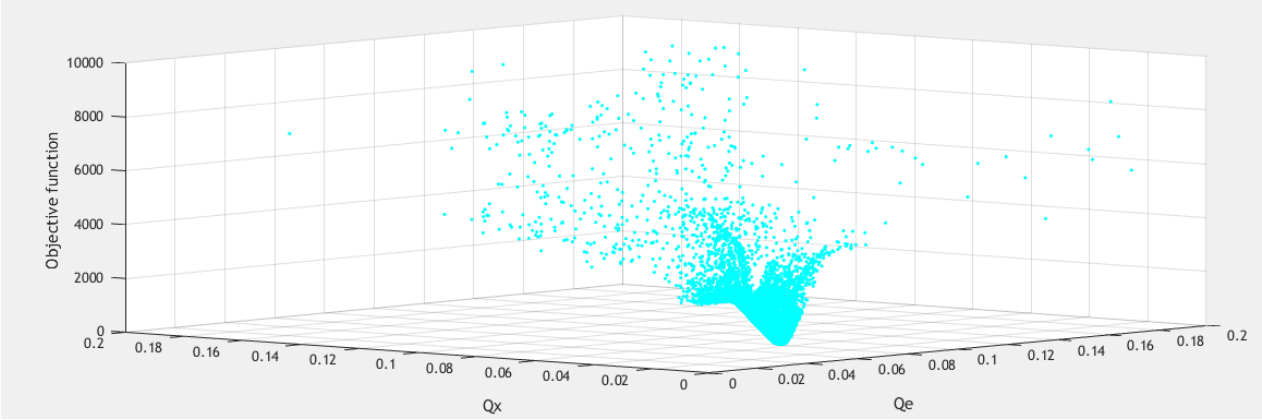

Figure 8.1 Parabolic form in Run9 of *Optimization 7.1* particles plot (flow-rates in mL/min).

Approaching this figure, the parabola is clearly visible.

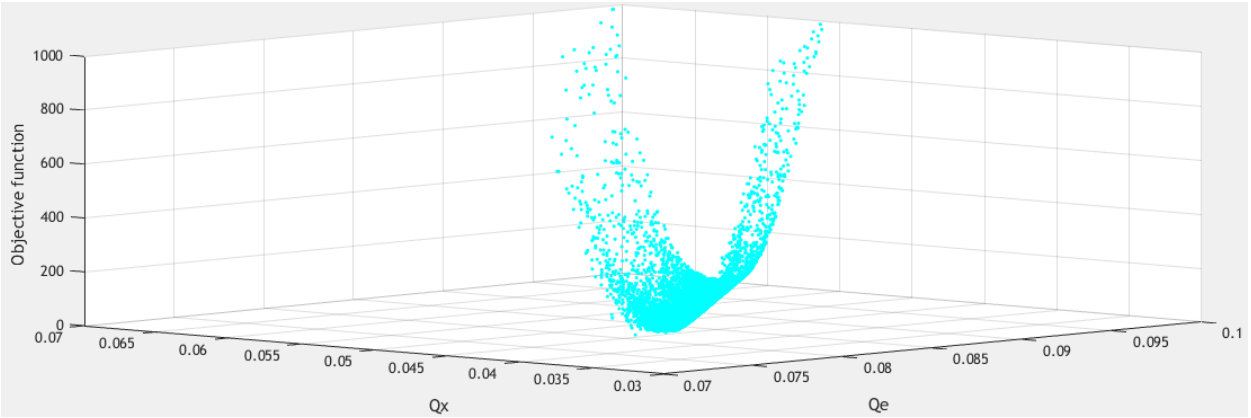

Figure 8.2 Parabolic form in Run9 of *Optimization 7.1* particles plot (flow-rates in mL/min).

Figure 8.2 shows that there is range of points in the optimum region (a line can almost be passed through those points); moreover, in the previous figures the optimum point is not visible which indicates it might be in the other side. The figure was then switched.

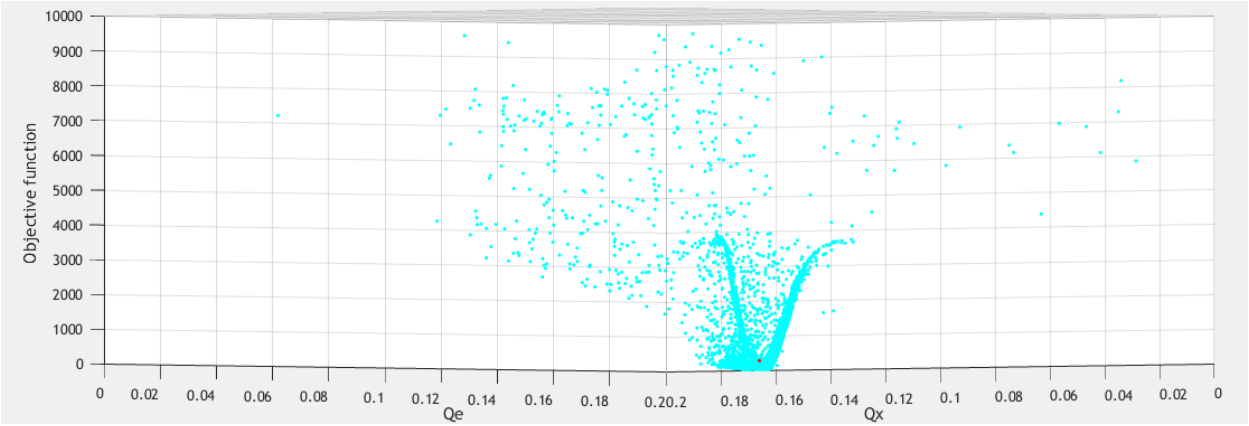

Figure 8.3 Parabolic form in Run9 of *Optimization 7.1* particles plot (flow-rates in mL/min).

Now the optimum point is represented and approaching Figure 8.3 it is visible that it is in the parabola's vertex (Figure 8.4).

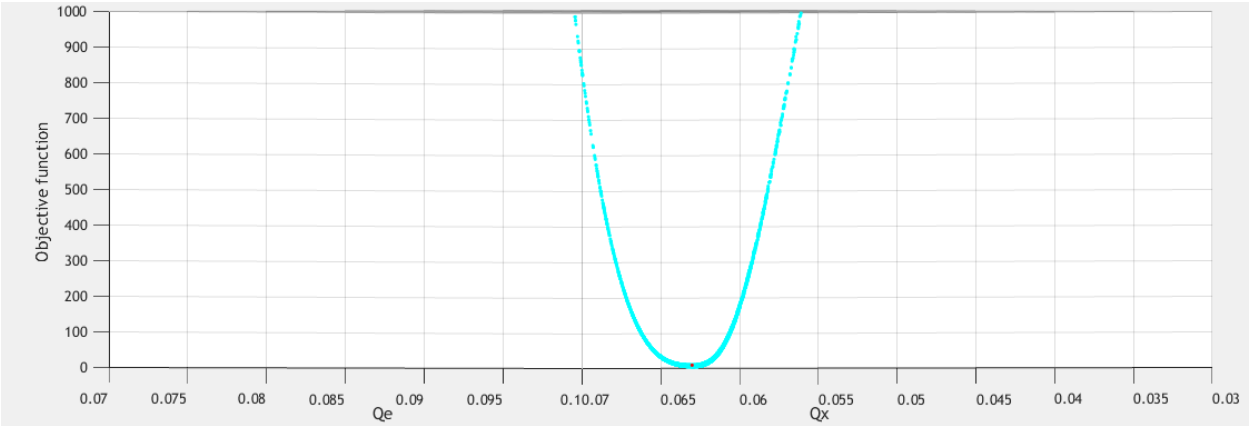

Figure 8.4 Parabolic form in Run9 of *Optimization 7.1* particles plot (flow-rates in mL/min).

The penalty coefficient ω leads to the creation of plateaus. For example, in the *Parallel* PSO optimization, ω was set to 4000 and 5000 for the productivity and the eluent consumption-based objective functions, respectively, which results in several plateaus that separate the feasible and the non-feasible regions.

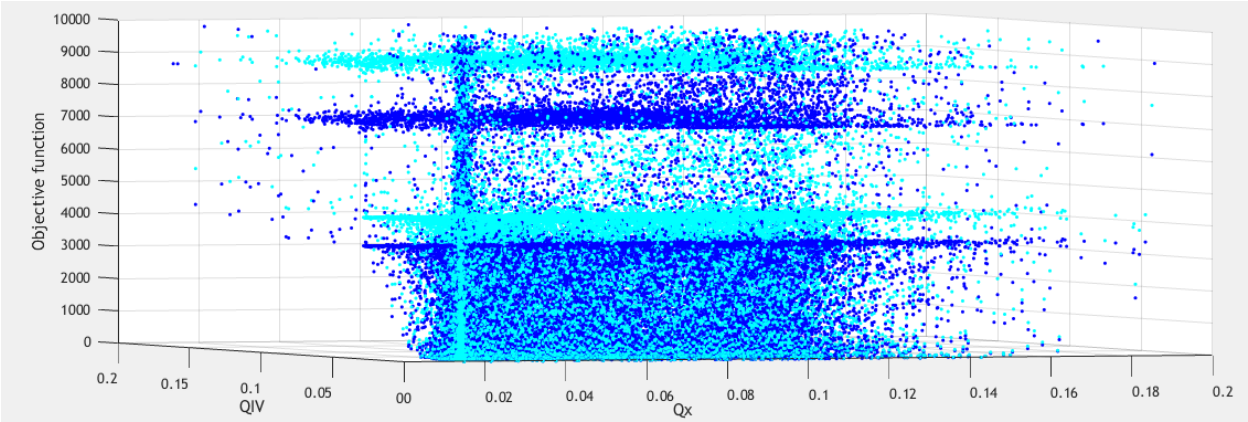

Figure 8.5 Plateaus in Run6 of *Optimization 5.7* particles plot (flow-rates in mL/min; $fobj_P$ in light blue; $fobj_E$ in dark blue).

Around a value of 4000 for the objective function, it is visible a plateau that represents the particles that led to purities slightly lower than 97%. The plateaus at 7000 and 9000 represent the ones that have values that are further away from 97%. The less filled spaces between the plateus indicate that those regions are non-feasible, i.e., the particles that would eventualy lead to those values of objective function are not physically possible.

To avoid points that are not physically possible, e.g., sets of flow-rates that lead to negative section flow-rates, a constraint was imposed in the algorithm to prevent the point from entering in the objective function, which was immediately set to 80000 as shown by the plateau in Figures 8.6 and 8.7.

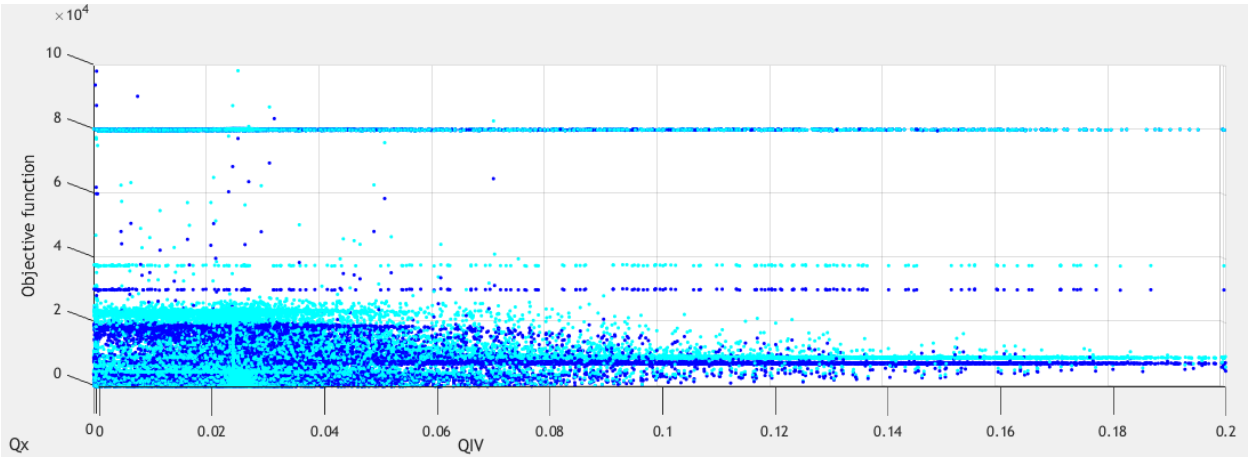

Figure 8.6 Plateaus in Run6 of *Optimization 5.7* particles plot (flow-rates in mL/min).

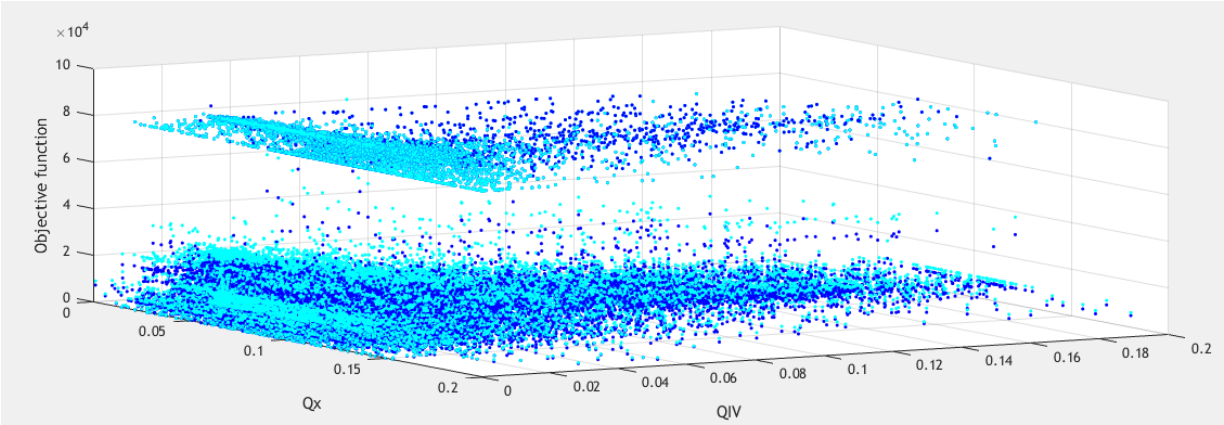

Figure 8.7 Plateaus in Run6 of *Optimization 5.7* particles plot (flow-rates in mL/min).

In another optimization which had the objective function value for the particles that were not physically possible set to 40000 the regions are also visible.

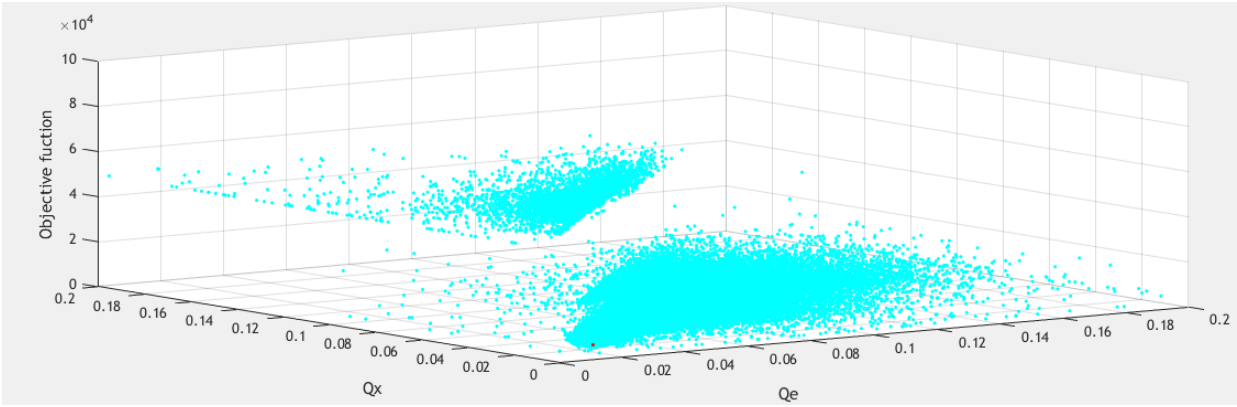

Figure 8.7 Regions in Run4 of *Optimization 5.4* particles plot (flow-rates in mL/min).

The points around 40000 are not physically possible so the algorithm changes them until they reach better values; they will then be on the right side of the previous figure.

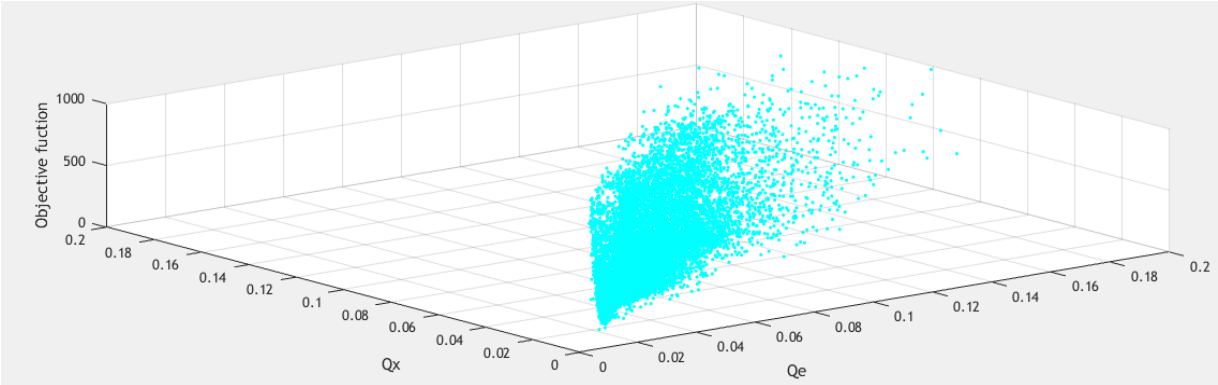

Figure 8.8 Right side regions in Run4 of *Optimization 5.4* particles plot (flow-rates in mL/min).

Which, as already seen, assumes a parabolic form.

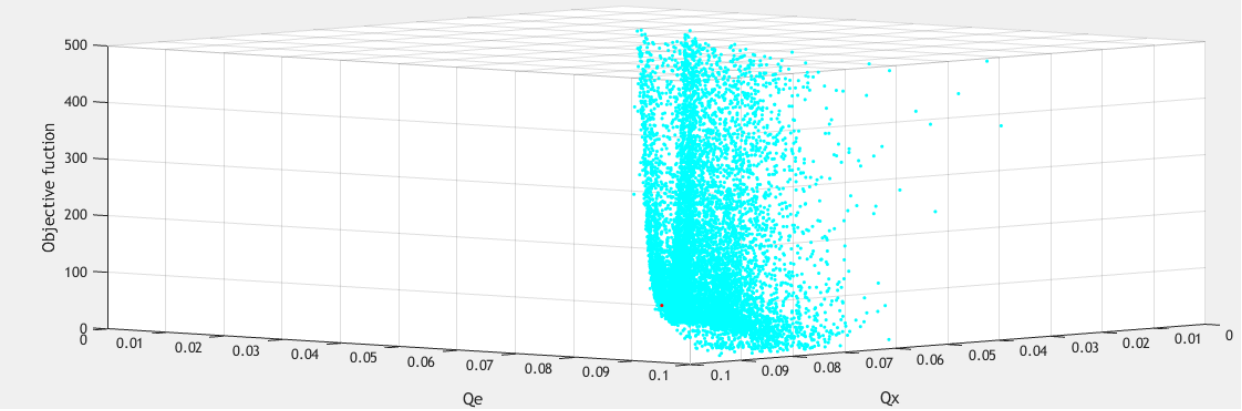

Figure 8.9 Parabolic form in Run4 of *Optimization 5.4* particles plot (flow-rates in mL/min).

The next figure shows the "path" made for some particles, during the optimization.

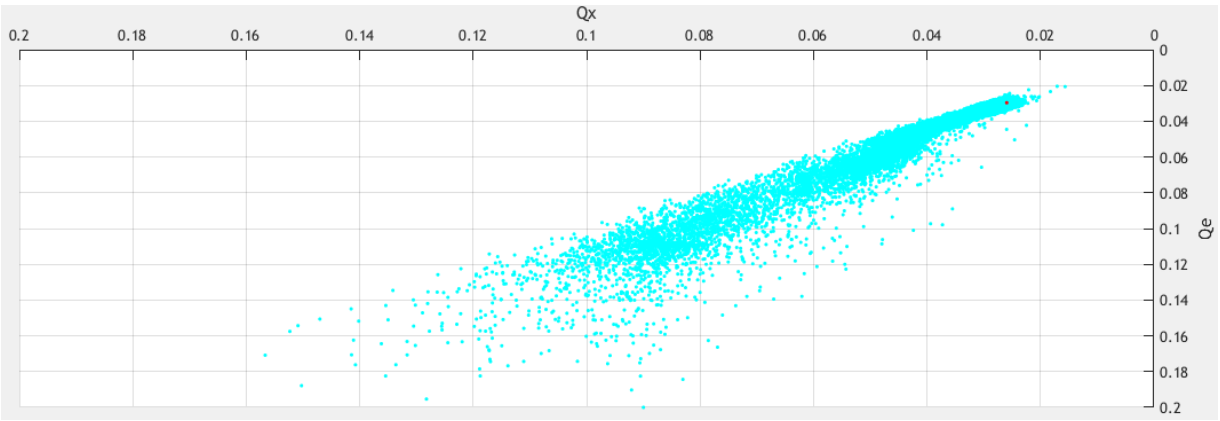

Figure 8.10 Optimization "path" in Run4 of *Optimization 5.4* particles plot (flow-rates in mL/min).

9 Summary

In a general way, the PSO algorithm is a recommended method to optimize the kind of system considered in this work, since the obtained optimum points were better than the ones already published. The purity constraints were respected and a higher value of productivity was achieved.

Concerning the PSO characteristics, the PSO parameters, *c1*, *c2* and *w* have a great impact on the optimization. The optimization processes lead to the creation of plateaus which represent the break-up of the zones where the separation is feasible from the non-feasible ones.

The *Parallel* PSO, is a new variant of the original PSO in which the optimization variables are spread and optimized using two distinct objective functions, simultaneously. This method showed great results.

Regarding the objective functions, a productivity-based objective function can be used to optimize the feed and the solid flow-rates. On the other hand, to optimize the eluent, extract and recycle flow-rates, the eluent consumption needs to be included in the objective function. An objective function that comprises both the productivity and the eluent consumption leads to a better result than the *Parts optimization*. The inclusion of a penalty is a clever way to restrain the purities. The objective functions values progress in a parabolic form.

The CPU time naturally increases as the number of optimization variables or iterations increases. Likewise, the number of iterations raises with the number of properties to optimize.

Looking to the TMB model, the extract flow-rate increases with the eluent flow-rate and the recycle flow-rate is related with the previous ones. A larger feed flow-rate leads to higher productivity. If the eluent flow-rate is not limited, it tends to increase in order to better regenerate the solid. The TMB configuration exerts a crucial influence on the separation. In fact, the productivity was highly increased (50% higher than the results reported in the literature; Wu et al., 2006) when the section's lengths were allowed to variate.

As the number of columns of the SMB increases, this model approaches the theoretical model, the TMB. A fourteen column SMB allowed to achieve similar results to those obtained with the TMB.

The Varicol model allows to have a different number of columns per section which is a way to reduce the number of columns without compromising significantly the productivity.

As an accomplishement of this work, a paper was submitted to Industrial & Engineering Chemistry Research.

9.1 Suggestions for future work

As a continuation of this work, the following suggestions are made.

Due to its importance, explore the *c1*, *c2* and *w* parameters with the purpose of achieving new equations adapted to this specific system should be an interesting study. Moreover, in section 5.1.2 several formulas for the calculation of *w* were presented; the ones that were not tested could be.

In this work, the values for the penalty coefficient were determined by test runs. For the *Single optimization*, it would be advantageous to do a cost study to better balance the weights of the eluent consumption and the productivity.

Although the TMB acceptably represents the SMB (using a certain number of columns), the results are not the same. It would then be worth to use the PSO method with the SMB model itself.

Following the previous line, once the TMB configuration had such an impact, the configuration of the SMB (length and number of columns per section) should be included in the optimization.

The results obtained with the SMB model did not respect the purities constraints, some changes to the operating conditions such as reducing the feed or increasing the number of columns could be made in order to improve the result.

The Varicol is a complex model since more than one configuration is admitted, a more detailed study should then be done with the aim of achieving better results, using this technology.

10 References

Al-Hassan, W., Fayek, M.B., Shaheen, S.I. “Psosa: An Optimized Particle Swarm Technique for Solving the Urban Planning Problem”. *Computer Engineering And Systems*, 401–405 (2006).

Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S. Abraham, A. “Inertia Weight Strategies in Particle Swarm”. *Third World Congress on Nature and Biologically Inspired Computing*, 640–47 (2011).

Bentley, J., Sloan, C., Kawajiri, Y. “Simultaneous Modeling and Optimization of Nonlinear Simulated Moving Bed Chromatography by the Prediction – Correction Method”. AICHE, 1280, 51–63 (2013).

Broughton, D.B., Gerhold, C.G. Continuous Sorption Process Employing Fixed Bed of Sorbent and Moving Inlets and Outlets. US Patent 2 985 589 (1961).

Carson, D.B, Purse, F.V .Rotary Valve. US Patent 3 040 777 (1962).

Chen, G., Huang, X., Jia, J., Min, Z. “Natural Exponential Inertia Weight Strategy in Particle Swarm Optimization”. *Intelligent Control and Automation*, 3672–75 (2006).

Clerc, M. *Particles Swarm Optimization*. ISTE, Great Britain and United States, 2006.

Day, D.T.“A Suggestion as to the Origin of Pennsylvania Petroleum”. In *Proceedings of the American Philosophical Society Held at Philadelphia for Promoting Useful Knowledge*, 112–15 (1897).

Eberhart, R.C., Shi, Y. “Tracking and Optimizing Dynamic Systems with Particle Swarms”. *Evolutionary Computation* ,1, 94–100 (2001).

Eberhart, R.C., Shi, Y. “Particle Swarm Optimization: Developments, Applications and Resources”. *Proceedings of the 2001 Congress on Evolutionary Computation,* 1, 81–86 (2001).

Eberhart, R.C, Kennedy, J. “A New Optimizer Using Particle Swarm Theory”. *Proceedings of the Sixt Internation Symposium on Micro Machine and Human Science*, 1942-1948 (1995).

Faria, R.P.V, Rodrigues, A.E. "Instrumental aspects of Simulated Moving Bed chromatography". *Joural of Chromatography A*, 1421, 82-102 (2015).

Feng, Y., Teng, G.F., Wang, X., Yao, Y.M. “Chaotic Inertia Weigth in Particles Swarm Optimization”. *Innovative Computing Information and Control* (2007).

Gamwell, L. *Mathematics and Art: A Cultural History*. Princeton University Press,2016.

Gao, Y., An, X., Liu, J. “Particle Swarm Optimization Algorithm with Logarithm Decreasing Inertia Weight and Chaos Mutation”. *Computational Intelligence and Security*, 61–65 (2008).

Gomes, P., Minceva, M., Rogrigues, A.E. “Simulated moving bed technology:old and new”. *Adsortion*, 12, 375-392 (2006).

Heppner, F., Grenander, U. *A Stochastic Nonlinear Model for Coordinated Bird Flocks*. AAAS, Washington DC, 1990.

Kentzoglanakis, K., Poole, M. “Particle Swarm Optimization with an Oscillating Inertia Weight”. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 1749–50 (2009).

Li, H.R., Gao, Y.L.“Particle Swarm Optimization Algorithm with Exponent Decreasing Inertia

Weight And Stochastic Mutation". *Second International Conference on Information and Computing Science, 6*6–69 (2009).

Lorenz, H., Capla, F., Polenske, D., Elsner, M.P., Seidel-Morgenstern, A. "Crystallization Based Separation of Enantiomers". *Journal Chemical Engineering & Processing*, 42(4510), 5–16 (2007).

Ludemann-Hombourger, O., Nicoud, R.M., Bailly, M. "The 'VARICOL' Process: A New Multicolumn Continuous Chromatographic Process". *Separation Science and Technology*, 35(12), 1829 –1862 (2000).

Luís Manuel Santos Pais."Chiral Separation by Simulated Moving Bed Chromatography". Phd Thesis of University of Porto (1999).

Nogueira, I., Ribeiro, A., Rodrigues, A., Loureiro, J. "Effect of Operating Variables on Performance Indicators Using Orthogonalization Method". *Computers and Chemical Engineering*, 86, 5–7 (2016).

Pais, L.S., Loureiro, J. M., Rodrigues, A.E. "Modeling Strategies for Enantiomers Separation by SMB Chromatography". AIChE, 44 (3), 561–569 (1998).

Pálovics, E., Faigl, F., Fogassy, E. "Separation of the Mixtures of Chiral Compounds by Crystallization". *Advances in Crystallization Processes*, 3 (2012).

Ratnaweera, A., Halgamuge, S.K., Watson, H.C. "Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients". *Transactions on Evolutionary Computation*, 8(3), 240–255 (2004).

Reynolds, C.W. "Flocks, Herds and Schools: A Distributed Behavioral Model". *Computer Graphics*, 25–34 (1987).

Rodrigues, A.E., Pereira, C., Minceva, M., Pais, L.S., Ribeiro, A., Ribeiro, A., Silva, M., Graça, N., Santos, J.C. *Simulated Moving Bed Technology*. Butterworth-Heinemann, 2015.

Sha, D. Y., Lin, H. "Expert Systems with Applications: A Multi-Objective PSO for Job-Shop Scheduling Problems". *Expert Systems With Applications*, 37(2), 1065–1070 (2009).

Shi, Y., Eberhart, R.C. "A Modified Particle Swarm Optimizer". *Evolutionary Computation Proceedings*, 67–73 (1998).

Toumi, A., Engell, S., Diehl, M., Bock, H.G., Schl, J. 2007. "Efficient Optimization of Simulated Moving Bed Processes". *Chemical Engineering and Processing*, 46, 1067–84 (2006).

Tswett, M.S. *Chlorophylls in the Plant and Animal World,* 1910.

Wilson, E.O. *Sociobiology: The New Synthesis*. Harvard University Press, Cambridge, Massachusetts and London, England, 1975.

Wu, X., Jin, X., Xu, Z., Wang, S. "Application of Particle Swarm Optimization in Process of Non-Linear Simulated Moving Bed Chromatographic Fractionation". *Control and Instruments in Chemical Industry*, 33, 5–9 (2006).

Zhang, Z., Hidajat, K., Ray, A.J., Morbidelli, M. "Multiobjective Optimization of SMB and Varicol Process for Chiral Separation". AICHE, 48 (12) (2002).

Appendix A

Listings of PSO algorithms and called programs codes

A.1 PSO algorithm

```
%                              PSO ALGORITHM

%Needed programs
%
    Fobj
%    PSOTMB

%INPUT VARIABLES
%w0   -initial inertia weight
%wf   -final inertia weight
%nd   -number of dimensions
%np   -number of particles
%nit  -number of iterations
%xmin -vector of the minimum limits of the parameters
%xmax -vector of maximum limits of the parameters
%c1   -parameter
%c2   -parameter

%Share vector p
global p
%Start the communication with gPROMS
gOMATLAB('start','PSOTMB','PSOTMB','PSOTMB');
%Start time counting
tic
%Set parameters
w0=0.9;
wf=0.4;
nd=5; %8 if TMB configuration included
np=50;
nit=2000;
% x=[Qe Qx Q4 Qfeed Qs]
xmin=[0.01e-3 0.01e-3 0.01e-3 0.01e-3 0.01e-3];
xmax=[200e-3 200e-3 200e-3 200e-3 200e-3];
%if TMB configuration included
%LL=4*2.1;
%xmin=[0.01E-3 0.01E-3 0.01E-3 0.01E-3 0.01E-3 0.01*LL 0.01*LL 0.01*LL];
%xmax=[200E-3 200E-3 200E-3 200E-3 200E-3 0.36*LL 0.36*LL 0.36*LL];

%Reinitialize random algorithm
rng('shuffle');

%Initialize particles system
for k=1:nd
    vmax(k)=(xmax(k)-xmin(k))/5;
         for j=1:np
             xval(1,j,k)=xmin(k)+rand(1)*(xmax(k)-xmin(k));
             v(1,j,k)=vmax(k)*(2*rand(1)-1);
         end
end

%Start optimization
contador=1
%z=0.35;
for i=1:nit
%   CALCULATION OF w (chose only ONE of the cases)
%   Case one
    w=w0+(wf-w0)*(i/nit);
%   Case two
%   if nit < 500
```

```
%       w=w0+(wf-w0)*(i/500);
%    else
%       w=w0+(wf-w0)*(1);
%    end
%    Case three
%    w=0.5+rand(1)/2;
%    Case four (activate z=0.35)
%    w=(w0-wf)*((nit-i)/nit)+wf*z;
%    z=4*z*(1-z);

%    CALCULATION OF c1 AND c2 (chose only ONE of the cases)
%    Case one
     c1=(0.5-2.5)*(i/nit)+2.5;
     c2=(2.5-0.5)*(i/nit)+0.5;
%    Case two
%    if nit < 500
%       c1=(0.5-2.5)*(i/500)+2.5;
%       c2=(2.5-0.5)*(i/500)+0.5;
%    else
%       c1=(0.5-2.5)*(1)+2.5;
%       c2=(2.5-0.5)*(1)+0.5;
%    end

%     Evaluation of the objective function
      for j=1:np
      q1=xval(i,j,3)+xval(i,j,1);
      q2=q1-xval(i,j,2);
      q3=q2+xval(i,j,4);
      qr=-xval(i,j,3)+q3;
%     L4=LL-xval(i,j,6)-xval(i,j,7)-xval(i,j,8); if TMB configuration included
           if abs(qr)+abs(q1)+abs(q2)+abs(q3)==qr+q1+q2+q3
%           if TMB configuration included:
%           if abs(qr)+abs(q1)+abs(q2)+abs(q3)==qr+q1+q2+q3 & L4>0
             F(i,j)=fobj(xval(i,j,:));
          else
             F(i,j)=80000;
          end
      end

%     Best of each particle until the current iteration
      for k=1:nd
          for j=1:np
              [Fbest(j),pbest(j)]=min(F(:,j));
              xpbest(j,k)=xval(pbest(j),j,k);
          end
      end

%     Best particle until the current iteration
      [FF(i),ppbest(i)]=min(F(i,:));
      for k=1:nd
          xbest(i,k)=xval(i,ppbest(i),k);
          a=np*i;
          FFF=reshape(F',a,1);
          FFbest(i)=min(FFF);
          if i==1
             xgbest(i,k)=xbest(i,k)
          end
          if i>1
             if FF(i)>FFbest(i)
                 xgbest(i,k)=xgbest(contador,k)
             else
                 xgbest(i,k)=xbest(i,k)
```

```
                contador=i;
            end
        end
    end

    for k=1:nd
        for j=1:np
%           Update "step"
            v(i+1,j,k)=w*v(i,j,k)+c1*rand(1)*(xpbest(j,k)
            - xval(i,j,k))+c2*rand(1)*(xgbest(i,k)-xval(i,j,k));
            if abs(v(i+1,j,k))> vmax(k);
               v(i+1,j,k)=vmax(k)*sign(v(i+1,j,k));
            end
%           Update particle's position
            xval(i+1,j,k)=xval(i,j,k)+v(1+i,j,k);
            if xval(i+1,j,k)>xmax(k);
               xval(i+1,j,k)=xmax(k);
            end
            if xval(i+1,j,k)<xmin(k);
               xval(i+1,j,k)=xmin(k);
            end
        end
    end
end

%Final solution
for k=1:nd
   xxgbest(k)=xgbest(i,k)
   ybest=fobj(xgbest(nit,:))
end

%Stop communication with gPROMS
gOMATLAB('stop');
%Calculate CPU
toc
CPU=toc

%% PLOT
 contador=1;
 for i=1:nit
     for j=1:np
         for k=1:nd
             for k=1:nd
                 xb(contador,k)= xval(i,j,k);
             end
             contador=contador+1;
         end
     end
 end

 for k=1:nd-1
     figure(k)
     plot(xb(:,k),xb(:,k+1),'.b')
     hold on
     plot(xxgbest(k),xxgbest(k+1),'.r')
     xlabel(['Parameter',int2str(k)])
     ylabel(['Parameter',int2str(k+1)])
 end

 for k=1:nd-1
     figure(k+nd-1)
```

```
     plot3(xval(1:end-1,:,k),xval(1:end-1,:,k+1),F(:,:),'.b')
     hold on
     plot3(xxgbest(k),xxgbest(k+1),ybest,'.r')
     xlabel(['Parameter',int2str(k)])
     ylabel(['Parameter',int2str(k+1)])
     zlabel('Objective function)
 end
```

A.2 Objective functions

A.2.1 Productivity-based objective function

```
function min=fobj(x)
    %a=pr ;b=px ;c=recr ;d=recx
    % Qe Qx Q4 Qfeed Qs
    %pr px prodx prodr
    global p
    p = gOMATLAB('evaluate',[x(1) x(2) x(3) x(4) x(5)], 4);
    f1=(p(1)-0.97)-abs(p(1)-0.97);
    f2=(p(2)-0.97)-abs(p(2)-0.97);
    a=1/(1+p(3))+4000*(f1^2+f2^2);
    min=a;
end
```

A.2.2 Eluent consumption-based objective function

```
function min=fobj(x)
    %a=pr ;b=px ;c=recr ;d=recx
    % Qe Qx Q4 Qfeed Qs
    %pr px prodx prodr
    global p
    p = gOMATLAB('evaluate',[x(1) x(2) x(3) x(4) x(5)], 4);
    f1=(p(1)-0.97)-abs(p(1)-0.97);
    f2=(p(2)-0.97)-abs(p(2)-0.97);
    a1=p(4)+5000*(f1^2+f2^2);
    min=a1;
end
```

A.2.3 Productivity and eluent consumption-based objective function

```
function min=fobj(x)
    %a=pr ;b=px ;c=recr ;d=recx
    % Qe Qx Q4 Qfeed Qs
    %pr px prodx prodr
    global p
%   If TMB configuration included use
%   p = gOMATLAB('evaluate',[x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8)], 4);
   p = gOMATLAB('evaluate',[x(1) x(2) x(3) x(4) x(5)], 4);
    f1=(p(1)-0.97)-abs(p(1)-0.97);
    f2=(p(2)-0.97)-abs(p(2)-0.97);
    a=p(4)+1/(0.01+p(3))+4000*(f1^2+f2^2);
    if p(1)>=1
        p(1)=1;
    end
     if p(2)>=1
        p(2)=1;
    end
      if p(3)<=0
        p(3)=0;
    end
     if p(4)<=0
        p(4)=0;
    end
    min=a;
end
```

A.3 Parallel PSO

```
%                              PARALLEL PSO

%Needed programs
%    Fobjparallel
%    PSOTMB

%INPUT VARIABLES
%w0   -initial inertia weight
%wf   -final inertia weight
%nd   -number of dimensions
%np   -number of particles
%nit  -number of iterations
%xmin -vector of the minimum limits of the parameters
%xmax -vector of maximum limits of the parameters
%c1   -parameter
%c2   -parameter

%Share vector p
global p
%Start the communication with gPROMS
gOMATLAB('start','PSOTMB','PSOTMB','PSOTMB');
%Start time counting
tic
%Set parameters
w0=0.9;
wf=0.4;
nd=5 ;
np=50;
nit=2000 ;
% x=[Qe Qx Q4 Qfeed Qs]
xmin=[0.01e-3 0.01e-3 0.01e-3 0.01e-3 0.01e-3] ;
xmax=[200e-3 200e-3 200e-3 200e-3 200e-3] ;
%Reinitialize random algorithm
rng('shuffle');

%Initialize particles system
for k=1:nd
    vmax(k)=(xmax(k)-xmin(k))/5 ;
         for j=1:np
             xval(1,j,k)=xmin(k)+rand(1)*(xmax(k)-xmin(k)) ;
             if k>=1 && k<=3
             ve(1,j,k)=vmax(k)*(2*rand(1)-1) ;
             else
             vp(1,j,k)=vmax(k)*(2*rand(1)-1) ;
             end
         end
end

%Start optimization
contador=1
contador1=1
for i=1:nit
%    Calculation of w, c1 and c2
     w=w0+(wf-w0)*(i/nit);
     c1=(0.5-2.5)*(i/nit)+2.5;
     c2=(2.5-0.5)*(i/nit)+0.5;
```

```
%    Evaluation of the objective function
     for j=1:np
         qr=xval(i,j,4)+xval(i,j,1)-xval(i,j,2);
         q1=xval(i,j,3)+xval(i,j,1);
         q2=q1-xval(i,j,2);
         q3=q2+xval(i,j,4);
         qr=-xval(i,j,3)+q3;
         if abs(qr)+abs(q1)+abs(q2)+abs(q3)==qr+q1+q2+q3
            fun= fobj(xval(i,j,:));
            Fprod(i,j)=fun(1);
            Felu(i,j)=fun(2);
          else
             Fprod(i,j)=80000;
             Felu(i,j)=80000;
          end
     end

%    Best of each particle until de current iteration
     for k=1:3
         for j=1:np
             [Fbeste(j),pbeste(j)]=min(Felu(:,j));
             xpbeste(j,k)=xval(pbeste(j),j,k);
         end
     end

     for k=4:5
         for j=1:np
             [Fbestp(j),pbestp(j)]=min(Fprod(:,j));
             xpbestp(j,k)=xval(pbestp(j),j,k);
         end
     end

%   Best particle until the current itaration
    [FFp(i),ppbestp(i)]=min(Fprod(i,:));
    [FFe(i),ppbeste(i)]=min(Felu(i,:));

     for k=1:3
         xbeste(i,k)=xval(i,ppbeste(i),k);
         FFbeste(i)=min(FFe);
         if i==1
            xgbeste(i,k)=xbeste(i,k)
         end
         if i>1
            if FFe(i)>FFbeste(i)
                xgbeste(i,k)=xgbeste(contador,k)
            else
                xgbeste(i,k)=xbeste(i,k)
                contador=i;
            end
         end
     end

     for k=4:5
         xbestp(i,k)=xval(i,ppbestp(i),k);
         FFbestp(i)=min(FFp);
         if i==1
            xgbestp(i,k)=xbestp(i,k)
         end
         if i>1
            if FFp(i)>FFbestp(i)
                xgbestp(i,k)=xgbestp(contador1,k)
            else
```

```
                xgbestp(i,k)=xbestp(i,k)
                contador1=i;
            end
        end
    end

    for k=1:nd
        for j=1:np
%           Update "step"
            if k>=1 && k<=3
               ve(i+1,j,k)=w*ve(i,j,k)+c1*rand(1)*(xpbeste(j,k)
               -xval(i,j,k))+c2*rand(1)*(xgbeste(i,k)-xval(i,j,k));
               if abs(ve(i+1,j,k))> vmax(k);
                  ve(i+1,j,k)=vmax(k)*sign(ve(i+1,j,k));
               end
            else
               vp(i+1,j,k)=w*vp(i,j,k)+c1*rand(1)*(xpbestp(j,k)
               -xval(i,j,k))+c2*rand(1)*(xgbestp(i,k)-xval(i,j,k));
                  if abs(vp(i+1,j,k))> vmax(k);
                     vp(i+1,j,k)=vmax(k)*sign(vp(i+1,j,k));
                  end
            end

%           Update particle's position
            if k>=1 && k<=3
                xval(i+1,j,k)=xval(i,j,k)+ve(1+i,j,k);
                if xval(i+1,j,k)>xmax(k);
                   xval(i+1,j,k)=xmax(k);
                end
                if xval(i+1,j,k)<xmin(k);
                   xval(i+1,j,k)=xmin(k);
                end
            else
                xval(i+1,j,k)=xval(i,j,k)+vp(1+i,j,k);
                if xval(i+1,j,k)>xmax(k);
                   xval(i+1,j,k)=xmax(k);
                end
                if xval(i+1,j,k)<xmin(k);
                   xval(i+1,j,k)=xmin(k);
                end
            end
        end
    end
end

%Final solution
%ELUENT
%[min de cada particula, iteraçao]
 [Fbestee,itbestee]=min(Felu);
%[min dos min, particula]
 [besteeee,pbesteeee]=min(Fbestee);
%it
 ite=itbestee(pbesteeee);
 for k=1:nd
     xpbesteee(k)=xval(itbestee(pbesteeee),pbesteeee,k);
 end
 yelu=fobj(xpbesteee) ;
 pelu=p;
 save('pelu','pelu');
 %PRODUCTIVITY
%[min de cada particula, iteraçao]
 [Fbestpp,itbestpp]=min(Fprod);
```

```
%[min dos min, particula]
 [bestpppp,pbestpppp]=min(Fbestpp);
%it
 itp=itbestpp(pbestpppp);
 for k=1:nd
     xpbestppp(k)=xval(itbestpp(pbestpppp),pbestpppp,k);
 end
 yprod=fobj(xpbestppp);
 pprod=p;
 save('pprod','pprod');

%Stop comumnication with gPROMS
   gOMATLAB('stop');
%Calculation of CPU
 toc
 CPU=toc

%% Plot
 contador=1;
 for i=1:nit
     for j=1:np
         for k=1:nd
             xb(contador,k)= xval(i,j,k);
         end
         contador=contador+1;
     end
 end

 for k=1:nd-1
     figure(k)
     plot(xb(:,k),xb(:,k+1),'.b')
     hold on
     plot(xxgbest(k),xxgbest(k+1),'.r')
     xlabel(['Parameter',int2str(k)])
     ylabel(['Parameter',int2str(k+1)])
 end

 for k=1:nd-1
     figure(nd-1+k)
     plot3(xval(1:end-1,:,k),xval(1:end-1,:,k+1),Fprod(:,:),'.b')
     hold on
     plot3(xval(1:end-1,:,k),xval(1:end-1,:,k+1),Felu(:,:),'.c')
     hold on
     plot3(xxgbest(k),xxgbest(k+1),ybest,'.r')
     xlabel(['Parameter',int2str(k)])
     ylabel(['Paramater',int2str(k+1)])
     zlabel('Objective function')
 end
```

A.3.1 Parallel objective function

```
function min=fobjparallel(x)
    %a=pr ;b=px ;c=recr ;d=recx
    % Qe Qx Q4 Qfeed Qs
    %pr px prodx prodr
    global p
    p = gOMATLAB('evaluate',[x(1) x(2) x(3) x(4) x(5)], 4);
    f1=(p(1)-0.97)-abs(p(1)-0.97);
    f2=(p(2)-0.97)-abs(p(2)-0.97);
    a=1/(1+p(3))+4000*(f1^2+f2^2);
    a1=p(4)+5000*(f1^2+f2^2);
```

```
    min=[a a1];
end
```

A.4 PSOTMB model

```
#                                PSOTMB PROCESS
#
#                  *uncomment if TMB configuration included*
#
# Needed programs
# functionPSOTMB

UNIT
PSOTMB as functionPSOTMB

SET
within PSOTMB do
    kl(1):=6;
    kl(2):=6;
    T:=303.15;
    pe:=2000 ;
    epsilon:=0.4;
    d:=0.26;
    n:=2 ;
    nj:=4;
    l:=2.1;
    A:=atan(1)*4*(d/2)^2 ;
    cfeed(1):=2.9;
    cfeed(2):=2.9;
    distancia:=[OCFEM,2,300];
    #LL:=4*2.1;
end

SOLUTIONPARAMETERS
ReportingInterval := 0.1
FPI := "eventFPI"

SCHEDULE
sequence
continue for 1
    save "System_State"
    while true do
        sequence
            restore "System_State"
            get
                PSOTMB.Qe;
                PSOTMB.Qx;
                PSOTMB.Q4;
                PSOTMB.Qfeed;
                PSOTMB.Qs;
#                PSOVARICOL.l(1);
#                PSOVARICOL.l(2);
#                PSOVARICOL.l(3);
#                PSOVARICOL.l(4);
            end
            continue for 2
            send
                PSOTMB.pr;
                PSOTMB.px;
                PSOTMB.P;
                PSOTMB.E;
          end
        end
    end
```

```
end
#                                functionPSOTMB
#
#                  *uncomment if TMB configuration included*

PARAMETER
A as real
l as real
n as integer
nj as integer
d as real
epsilon as real
T as real
pe as real
cfeed as array(n) of real
kl as array(n) of real
#LL as real

DISTRIBUTION_DOMAIN
distancia as [0:l]
#distancia as [0:1]

VARIABLE
Qfeed as valor
Qs as valor
Qe as valor
Qx as valor
Q4 as valor
pr as valor
px as valor
recr as valor
recx as valor
prodr as valor
prodx as valor
E as valor
P as valor
ur as valor
ue as valor
ux as valor
ufeed as valor
us as valor
u as array(nj) of valor
l as array(nj) of valor
dax as array(nj) of valor
cr as array(n) of valor
cx as array(n) of valor
c as distribution(n,nj,distancia) of valor
q as distribution(n,nj,distancia) of valor
qq as distribution(n,nj,distancia) of valor
ce as distribution (n,nj) of valor

 BOUNDARY
for j:=1 to nj do
    for i:=1 to n do
        c(i,j,0)-1/pe*partial(c(i,j,0),distancia)=ce(i,j);
    end
end

#ce(i,j) calculation
for i:=1 to n do
    c(i,4,1)=u(1)/u(4)*ce(i,1);
    c(i,1,1)=ce(i,2);
    c(i,2,1)=u(3)/u(2)*ce(i,3)-ufeed/u(2)*cfeed(i);
```

```
    c(i,3,1)=ce(i,4);
end

for i:=1 to n do
    q(i,4,1)=q(i,1,0);
    q(i,1,1)=q(i,2,0);
    q(i,2,1)=q(i,3,0);
    q(i,3,1)=q(i,4,0);
end

EQUATION
#section velocities
u(1)=u(4)+ue;
u(2)=u(1)-ux;
u(3)=u(2)+ufeed;
u(4)=u(3)-ur;

#axial dispersion coefficient calculation
for j:=1 to nj do
    dax(j)=u(j)*l(j)/pe;
end


# Liquid phase mass balance
for j:=1 to nj do
    for i:=1 to n do
        for k:=0|+ to l do
            0=dax(j)*partial(c(i,j,k),distancia,distancia)
            -u(j)*partial(c(i,j,k),distancia)-
            ((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
        end
    end
end
#for j:=1 to nj do
#    for i:=1 to n do
#        for k:=0|+ to 1 do
#            0=dax(j)/(l(j)*l(j))*partial(c(i,j,k),distancia,distancia)
             -u(j)/l(j)*partial(c(i,j,k),distancia)-
             ((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
#
#        end
#    end
#end


# Solid phase mass balance
for j:=1 to nj do
    for i:=1 to n do
        for k:=0 to l|- do
            0=kl(i)*(qq(i,j,k)-q(i,j,k))+us*partial(q(i,j,k),distancia);
        end
    end
end
#for j:=1 to nj do
#    for i:=1 to n do
#        for k:=0 to 1|- do
#            0=kl(i)*(qq(i,j,k)-
q(i,j,k))+us/l(j)*partial(q(i,j,k),distancia);
#        end
#    end
#end

# Equilibrium isotherms
```

```
for j:=1 to nj do
    for k:=0 to 1 do
        qq(1,j,k)=2.69*c(1,j,k)/(1+0.0336*c(1,j,k)
        +0.0466*c(2,j,k))+0.10*c(1,j,k)/(1+c(1,j,k)+3*c(2,j,k));
        qq(2,j,k)=3.73*c(2,j,k)/(1+0.0336*c(1,j,k)
        +0.0466*c(2,j,k))+0.30*c(2,j,k)/(1+c(1,j,k)+3*c(2,j,k));
    end
end
#for j:=1 to nj do
#    for k:=0 to 1 do
#        qq(1,j,k)=2.69*c(1,j,k)/(1+0.0336*c(1,j,k)
         +0.0466*c(2,j,k))+0.10*c(1,j,k)/(1+c(1,j,k)+3*c(2,j,k));
#        qq(2,j,k)=3.73*c(2,j,k)/(1+0.0336*c(1,j,k)
         +0.0466*c(2,j,k))+0.30*c(2,j,k)/(1+c(1,j,k)+3*c(2,j,k));
#    end
#end

# Raffinate and extract concentrations
for i:=1 to n do
    cx(i)=c(i,1,1);
    cr(i)=c(i,3,1);
end

#l(4)=LL-l(1)-l(2)-l(3)-l(4);
pr=c(1,3,1)/sigma(cr);
px=c(2,1,1)/sigma(cx);
recr=(ur*A*epsilon*c(1,3,1))/(Qfeed*cfeed(1));
recx=(Qx*c(2,1,1))/(Qfeed*cfeed(2));
#prodr=(recr*Qfeed*cfeed(1))/((1-epsilon)*A*LL);
#prodx=(recx*Qfeed*cfeed(2))/((1-epsilon)*A*LL);
prodr=(recr*Qfeed*cfeed(1))/((1-epsilon)*A*l*4);
prodx=(recx*Qfeed*cfeed(2))/((1-epsilon)*A*l*4);
P=prodr+prodx;
E=(Qe+Qfeed)/(ur*A*c(1,3,1)+Qx*c(2,1,1));
u(4)=Q4/A/epsilon ;
ue=Qe/A/epsilon ;
ux=Qx/A/epsilon ;
us=Qs/A/(1-epsilon) ;
ufeed=Qfeed/A/epsilon;

ASSIGN
Q4:=21.47e-3;
Qe:=22.11e-3;
Qx:=19.59e-3;
Qs:=9e-3;
Qfeed:=4.41e-3;
#l(1):=2.1;
#l(2):=2.1;
#l(3):=2.1;
```

Appendix B
Listings of TMB, SMB and Varicol models codes

B.1 Steady-State TMB model

```
#                                TMB PROCESS
# Needed programs
# functionTMB

UNIT
TMB as functionTMB

SET
within TMB do
    kl(1):=6;
    kl(2):=6;
    T:=303.15;
    pe:=2000 ;
    epsilon:=0.4;
    d:=0.26;
    n:=2 ;
    nj:=4;
    l:=2.1;
    A:=atan(1)*4*(d/2)^2 ;
    cfeed(1):=2.9;
    cfeed(2):=2.9;
    distancia:=[OCFEM,2,300];
end

SOLUTIONPARAMETERS
ReportingInterval := 0.1

SCHEDULE
continue for 4


#                                functionTMB
#
#                *uncomment if TMB configuration included*

PARAMETER
A as real
l as real
n as integer
nj as integer
d as real
epsilon as real
T as real
pe as real
cfeed as array(n) of real
kl as array(n) of real

DISTRIBUTION_DOMAIN
distanciia as [0:l]

VARIABLE
Qfeed as valor
Qs as valor
Qe as valor
Qx as valor
Q4 as valor
pr as valor
px as valor
recr as valor
```

```
recx as valor
prodr as valor
prodx as valor
E as valor
P as valor
ur as valor
ue as valor
ux as valor
ufeed as valor
us as valor
u as array(nj) of valor
l as array(nj) of valor
dax as array(nj) of valor
cr as array(n) of valor
cx as array(n) of valor
c as distribution(n,nj,distancia) of valor
q as distribution(n,nj,distancia) of valor
qq as distribution(n,nj,distancia) of valor
ce as distribution (n,nj) of valor

 BOUNDARY
for j:=1 to nj do
    for i:=1 to n do
        c(i,j,0)-1/pe*partial(c(i,j,0),distancia)=ce(i,j);
    end
end

#ce(i,j) calculation
for i:=1 to n do
    c(i,4,1)=u(1)/u(4)*ce(i,1);
    c(i,1,1)=ce(i,2);
    c(i,2,1)=u(3)/u(2)*ce(i,3)-ufeed/u(2)*cfeed(i);
    c(i,3,1)=ce(i,4);
end

for i:=1 to n do
    q(i,4,1)=q(i,1,0);
    q(i,1,1)=q(i,2,0);
    q(i,2,1)=q(i,3,0);
    q(i,3,1)=q(i,4,0);
end

EQUATION
#section velocities
u(1)=u(4)+ue;
u(2)=u(1)-ux;
u(3)=u(2)+ufeed;
u(4)=u(3)-ur;

#axial dispersion coefficient calculation
for j:=1 to nj do
    dax(j)=u(j)*l(j)/pe;
end


# Liquid phase mass balance
for j:=1 to nj do
    for i:=1 to n do
        for k:=0|+ to l do
            0=dax(j)*partial(c(i,j,k),distancia,distancia)
            -u(j)*partial(c(i,j,k),distancia)-
            ((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
        end
```

```
    end
end

# Solid phase mass balance
for j:=1 to nj do
    for i:=1 to n do
        for k:=0 to l|- do
            0=kl(i)*(qq(i,j,k)-q(i,j,k))+us*partial(q(i,j,k),distancia);
        end
    end
end

# Equilibrium isotherms
for j:=1 to nj do
    for k:=0 to l do
        qq(1,j,k)=2.69*c(1,j,k)/(1+0.0336*c(1,j,k)
        +0.0466*c(2,j,k))+0.10*c(1,j,k)/(1+c(1,j,k)+3*c(2,j,k));
        qq(2,j,k)=3.73*c(2,j,k)/(1+0.0336*c(1,j,k)
        +0.0466*c(2,j,k))+0.30*c(2,j,k)/(1+c(1,j,k)+3*c(2,j,k));
    end
end

# Raffinate and extract concentrations
for i:=1 to n do
    cx(i)=c(i,1,1);
    cr(i)=c(i,3,1);
end

pr=c(1,3,1)/sigma(cr);
px=c(2,1,1)/sigma(cx);
recr=(ur*A*epsilon*c(1,3,1))/(Qfeed*cfeed(1));
recx=(Qx*c(2,1,1))/(Qfeed*cfeed(2));
prodr=(recr*Qfeed*cfeed(1))/((1-epsilon)*A*l*4);
prodx=(recx*Qfeed*cfeed(2))/((1-epsilon)*A*l*4);
P=prodr+prodx;
E=(Qe+Qfeed)/(ur*A*c(1,3,1)+Qx*c(2,1,1));
u(4)=Q4/A/epsilon ;
ue=Qe/A/epsilon ;
ux=Qx/A/epsilon ;
us=Qs/A/(1-epsilon) ;
ufeed=Qfeed/A/epsilon;

ASSIGN
Q4:=21.47e-3;
Qe:=22.11e-3;
Qx:=19.59e-3;
Qs:=9e-3;
Qfeed:=4.41e-3;
```

B.2 Steady-State TMB model with variable sections length

```
#                                TMB PROCESS
# Needed programs
# functionTMB

UNIT
TMB as functionTMB

SET
within TMB do
    kl(1):=6;
    kl(2):=6;
    T:=303.15;
    pe:=2000 ;
    epsilon:=0.4;
    d:=0.26;
    n:=2 ;
    nj:=4;
    L1:=1.8;
    L2:=3;
    L3:= 3;
    L4:=0.6;
    A:=atan(1)*4*(d/2)^2 ;
    cfeed(1):=2.9;
    cfeed(2):=2.9;
    distancia1:=[OCFEM,2,100];
    distancia2:=[OCFEM,2,167];
    distancia3:=[OCFEM,2,167];
    distancia4:=[OCFEM,2,33];
end

SOLUTIONPARAMETERS
ReportingInterval := 0.1

SCHEDULE
continue for 4


#                                functionTMB
#
#                 *uncomment if TMB configuration included*

PARAMETER
A as real
L1 as real
L2 as real
L3 as real
L4 as real
n as integer
nj as integer
d as real
epsilon as real
T as real
pe as real
cfeed as array(n) of real
kl as array(n) of real

DISTRIBUTION_DOMAIN
Distancia1 as [0:L1]
Distancia2 as [0:L2]
```

```
Distancia3 as [0:L3]
Distancia4 as [0:L4]

VARIABLE
Qfeed as valor
Qs as valor
Qe as valor
Qx as valor
Q4 as valor
pr as valor
px as valor
recr as valor
recx as valor
prodr as valor
prodx as valor
E as valor
P as valor
ur as valor
ue as valor
ux as valor
ufeed as valor
us as valor
u as array(nj) of valor
l as array(nj) of valor
dax as array(nj) of valor
cr as array(n) of valor
cx as array(n) of valor
c1 as distribution(n,distancia1) of valor
q1 as distribution(n,distancia1) of valor
qq1 as distribution(n,distancia1) of valor
c2 as distribution(n,distancia2) of valor
q2 as distribution(n,distancia2) of valor
qq2 as distribution(n,distancia2) of valor
c3 as distribution(n,distancia3) of valor
q3 as distribution(n,distancia3) of valor
qq3 as distribution(n,distancia3) of valor
c4 as distribution(n,distancia4) of valor
q4 as distribution(n,distancia4) of valor
qq4 as distribution(n,distancia4) of valor
ce as distribution (n,nj) of valor

 BOUNDARY
for i:=1 to n do
    c(i,0)-dax(1)/u(1)*partial(c1(i,0),distancia1)=ce(i, 1);
end
for i:=1 to n do
    c(i,0)-dax(2)/u(2)*partial(c2(i,0),distancia2)=ce(i, 2);
end
for i:=1 to n do
    c(i,0)-dax(3)/u(3)*partial(c3(i,0),distancia3)=ce(i, 3);
end
for i:=1 to n do
    c(i,0)-dax(4)/u(4)*partial(c4(i,0),distancia1)=ce(i, 4);
end

#ce(i,j) calculation
for i:=1 to n do
    c1(i,L4)=u(1)/u(4)*ce(i,1);
    c2(i,L1)=ce(i,2);
    c3(i,L2)=u(3)/u(2)*ce(i,3)-ufeed/u(2)*cfeed(i);
    c4(i,L3)=ce(i,4);
end
```

```
for i:=1 to n do
    q1(i,L4)=q1(i,0);
    q2(i,L1)=q2(i,0);
    q3(i,L2)=q3(i,0);
    q4(i,L3)=q4(i,0);
end

EQUATION
#section velocities
u(1)=u(4)+ue;
u(2)=u(1)-ux;
u(3)=u(2)+ufeed;
u(4)=u(3)-ur;

#axial dispersion coefficient calculation
dax(1)=u(1)*L1/pe;
dax(2)=u(2)*L1/pe;
dax(3)=u(3)*L1/pe;
dax(4)=u(4)*L1/pe;

# Liquid phase mass balance
for i:=1 to n do
    for k:=0|+ to L1 do
        0=dax(1)*partial(c1(i,k),distancia1,distancia1)
        -u(1)*partial(c1(i,k),distancia1)-
        ((1-epsilon)/epsilon)*kl(i)*(qq1(i,k)-q1(i,k));
    end
end
for i:=1 to n do
    for k:=0|+ to L2 do
        0=dax(2)*partial(c2(i,k),distancia2,distancia2)
        -u(2)*partial(c2(i,k),distancia2)-
        ((1-epsilon)/epsilon)*kl(i)*(qq2(i,k)-q2(i,k));
    end
end
for i:=1 to n do
    for k:=0|+ to L3 do
        0=dax(3)*partial(c3(i,k),distancia3,distancia3)
        -u(3)*partial(c3(i,k),distancia3)-
        ((1-epsilon)/epsilon)*kl(i)*(qq3(i,k)-q3(i,k));
    end
end
for i:=1 to n do
    for k:=0|+ to L4 do
        0=dax(4)*partial(c4(i,k),distancia4,distancia4)
        -u(4)*partial(c4(i,k),distancia4)-
        ((1-epsilon)/epsilon)*kl(i)*(qq4(i,k)-q4(i,k));
    end
end

# Solid phase mass balance
for i:=1 to n do
    for k:=0 to L1|- do
        0=kl(i)*(qq1(i,k)-q1(i,k))+us*partial(q1(i,k),distancia1);
    end
end
for i:=1 to n do
    for k:=0 to L2|- do
        0=kl(i)*(qq2(i,k)-q2(i,k))+us*partial(q2(i,k),distancia2);
    end
end
for i:=1 to n do
    for k:=0 to L3|- do
```

```
        0=kl(i)*(qq3(i,k)-q3(i,k))+us*partial(q3(i,k),distancia3);
    end
end
for i:=1 to n do
    for k:=0 to L4|- do
        0=kl(i)*(qq4(i,k)-q4(i,k))+us*partial(q4(i,k),distancia4);
    end
end

# Equilibrium isotherms
for k:=0 to L1 do
    qq1(1,k)=2.69*c1(1,k)/(1+0.0336*c1(1,k)
    +0.0466*c1(2,k))+0.10*c1(1,k)/(1+c1(1,k)+3*c1(2,k));
    qq1(2,k)=3.73*c1(2,k)/(1+0.0336*c1(1,k)
    +0.0466*c1(2,k))+0.30*c1(2,k)/(1+c1(1,k)+3*c1(2,k));
end
for k:=0 to L2 do
    qq2(1,k)=2.69*c2(1,k)/(1+0.0336*c2(1,k)
    +0.0466*c2(2,k))+0.10*c2(1,k)/(1+c2(1,k)+3*c2(2,k));
    qq2(2,k)=3.73*c2(2,k)/(1+0.0336*c2(1,k)
    +0.0466*c2(2,k))+0.30*c2(2,k)/(1+c2(1,k)+3*c2(2,k));
end
for k:=0 to L3 do
    qq3(1,k)=2.69*c3(1,k)/(1+0.0336*c3(1,k)
    +0.0466*c3(2,k))+0.10*c3(1,k)/(1+c3(1,k)+3*c3(2,k));
    qq3(2,k)=3.73*c3(2,k)/(1+0.0336*c3(1,k)
    +0.0466*c3(2,k))+0.30*c3(2,k)/(1+c3(1,k)+3*c3(2,k));
end
for k:=0 to L4 do
    qq4(1,k)=2.69*c4(1,k)/(1+0.0336*c4(1,k)
    +0.0466*c4(2,k))+0.10*c4(1,k)/(1+c4(1,k)+3*c4(2,k));
    qq4(2,k)=3.73*c4(2,k)/(1+0.0336*c4(1,k)
    +0.0466*c4(2,k))+0.30*c4(2,k)/(1+c4(1,k)+3*c4(2,k));
end


# Raffinate and extract concentrations
for i:=1 to n do
    cx(i)=c(i,1,1);
    cr(i)=c(i,3,1);
end

pr=c(1,3,1)/sigma(cr);
px=c(2,1,1)/sigma(cx);
recr=(ur*A*epsilon*c(1,3,1))/(Qfeed*cfeed(1));
recx=(Qx*c(2,1,1))/(Qfeed*cfeed(2));
prodr=(recr*Qfeed*cfeed(1))/((1-epsilon)*A*l*4);
prodx=(recx*Qfeed*cfeed(2))/((1-epsilon)*A*l*4);
P=prodr+prodx;
E=(Qe+Qfeed)/(ur*A*c(1,3,1)+Qx*c(2,1,1));
u(4)=Q4/A/epsilon ;
ue=Qe/A/epsilon ;
ux=Qx/A/epsilon ;
us=Qs/A/(1-epsilon) ;
ufeed=Qfeed/A/epsilon;

ASSIGN
Q4:=21.47e-3;
Qe:=22.11e-3;
Qx:=19.59e-3;
Qs:=9e-3;
Qfeed:=4.41e-3;
```

B.3 SMB model

```
#                                SMB PROCESS
# Needed programs
# functionSMB

UNIT
SMB as functionSMB

SET
within SMB do
    tt:=5.5/x(1);
    stepc:=1;
    x:=[3,3,3,3];
    fator:=10;
    nt:=fator*sigma(x(1:4));
    nc:=sigma(x(1:4));
    kl(1):=6;
    kl(2):=6;
    T:=303.15;
    pe:=2000 ;
    epsilon:=0.4;
    d:=0.26;
    n:=2 ;
    nj:=4;
    l:=2.1/x(1);
    A:=atan(1)*4*(d/2)^2 ;
    cfeed(1):=2.9;
    cfeed(2):=2.9;
    distancia:=[OCFEM,2,300];
end

SOLUTIONPARAMETERS
ReportingInterval := 0.01

SCHEDULE
while SMB.contador<=SMB.nt do
      sequence
            continue for SMB.tt
            reassign
                    SMB.pr:=sigma(old(SMB.cr1))
                    /(sigma(old(SMB.cr1))+sigma(old(SMB.cr2)));
                    SMB.px:=sigma(old(SMB.cx2))
                    /(sigma(old(SMB.cx1))+sigma(old(SMB.cx2)));
                    SMB.recr:=old(SMB.ur)*SMB.A*SMB.epsilon
                    *sigma(old(SMB.cr1))
                    /(old(SMB.Qfeed)*old(SMB.cfeed(1)));
                    SMB.recx:=old(SMB.Qx)*sigma(old(SMB.cx2))
                    /(old(SMB.Qfeed)*SMB.cfeed(2));
                    SMB.prod:=(old(SMB.recr)*old(SMB.ur)
                    *SMB.A*SMB.epsilon
                    *sigma(old(SMB.cr1)))
                    /((1-SMB.epsilon)*SMB.A*SMB.l*SMB.nc)
                    +(old(SMB.recx)*old(SMB.Qx)*sigma(old(SMB.cx2)))
                    /((1-SMB.epsilon)*SMB.A*SMB.l*SMB.nc);
                    SMB.EC:=(old(SMB.Qe)+old(SMB.Qfeed))
                    /(old(SMB.Qfeed)*sigma(old(SMB.cfeed)));
             End
```

```
            parallel
                 if SMB.y1<SMB.nc then
                    reassign
                        SMB.y1:=old(SMB.y1)+SMB.stepc;
                    end
                 else
                    reassign
                        SMB.y1:=1;
                    end
                 end
                 if SMB.y2<SMB.nc then
                    reassign
                        SMB.y2:=old(SMB.y2)+SMB.stepc;
                    end
                 else
                    reassign
                        SMB.y2:=1;
                    end
                end
                 if SMB.y3<SMB.nc then
                    reassign
                        SMB.y3:=old(SMB2.y3)+SMB.stepc;
                    end
                 else
                    reassign
                        SMB.y3:=1;
                    end
                end
                 if SMB.y4<SMB.nc then
                    reassign
                        SMB.y4:=old(SMB.y4)+SMB.stepc;
                    end
                 else
                    reassign
                        SMB.y4:=1;
                    end
                end
            end

             reassign
                   SMB2.contador:=old(SMB.contador)+1;
                   SMB2.pr:=0;
                   SMB2.px:=0;
                   SMB2.recr:=0;
                   SMB2.recx:=0;
                   SMB2.prod:=0;
                   SMB2.EC:=0;
            end

                reinitial
                    SMB2.cx1,
                    SMB2.cx2,
                    SMB2.cr1,
                    SMB2.cr2
                with
                    SMB2.cx1=0;
                    SMB2.cx2=0;
                    SMB2.cr1=0;
                    SMB2.cr2=0;
                end
        end
end
```

```
#                              functionSMB
#
#                 *uncomment if TMB configuration included*

PARAMETER
fator as integer
stepc as integer
nt as integer
nc as integer
x as array(ns) of integer
tt as real
A as real
l as real
n as integer
nj as integer
d as real
epsilon as real
T as real
pe as real
cfeed as array(n) of real
kl as array(n) of real

DISTRIBUTION_DOMAIN
distancia as [0:l]

VARIABLE
Qfeed as valor
Qs as valor
Qe as valor
Qx as valor
Q4 as valor
pr as valor
px as valor
recr as valor
recx as valor
prodr as valor
prodx as valor
E as valor
P as valor
ur as valor
ue as valor
ux as valor
ufeed as valor
us as valor
y1 as valor
y2 as valor
y3 as valor
y4 as valor
u as array(nc) of valor
l as array(nc) of valor
dax as array(nc) of valor
contador as valor
aa as valor
bb as valor
cx1 as array (ns) of valor
cx2 as array (ns) of valor
cr1 as array (ns) of valor
cr2 as array (ns) of valor
c as distribution(n,nc,distancia) of valor
q as distribution(n,nc,distancia) of valor
qq as distribution(n,nc,distancia) of valor
ce as distribution (n,nc) of valor
```

```
BOUNDARY

#calculation of ce(i,j)
#Z of first column = L
for j:=1 to nc do
     for i:=1 to n do
         partial(c(i,j,l),distancia)=0;
     end
end

for j:=1 to nc do
    if j=y1 then
       for i:=1 to n do
           if j-1<1 then
              c(i,nc,l)=u(1)/u(4)*ce(i,j);
            else
                c(i,j-1,l)=u(1)/u(4)*ce(i,j);
            end
           c(i,j,0)-dax(1)/u(1)*partial(c(i,j,0),distancia)=ce(i,j);
       end
    else
        if (j>y1 and j<y1+x(1)) or j<y1+x(1)-nc then
            for i:=1 to n do
                 if j-1<1 then
                                c(i,nc,l)=ce(i,j);
                            else
                                c(i,j-1,l)=ce(i,j);
                            end
                c(i,j,0)-dax(1)/u(1)*partial(c(i,j,0),distancia)=ce(i,j);
            end
        else
            if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
                for i:=1 to n do
                    if j-1<1 then
                        c(i,nc,l)=ce(i,j);
                    else
                        c(i,j-1,l)=ce(i,j);
                    end
                    c(i,j,0)-dax(2)/u(2)*partial(c(i,j,0),distancia)=ce(i,j);
                    end
            else
                if j=y3 then
                    for i:=1 to n do
                        if j-1<1 then
                            c(i,nc,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                        else
                            c(i,j-1,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                        end
                        c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                        =ce(i,j);
                     end
                else
                    if (j>y3 and j<y3+x(3)) or j<y3+x(3)-nc  then
                        for i:=1 to n do
                            if j-1<1 then
                                c(i,nc,l)=ce(i,j);
                            else
                                c(i,j-1,l)=ce(i,j);
                            end
                         c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                         =ce(i,j);
                         end
                     else
```

```
                        #if (j>=y4 and j<=y4+x(4)) or j<y4+x(4)-nc then
                         for i:=1 to n do
                            if j-1<1 then
                                c(i,nc,l)=ce(i,j);
                            else
                                c(i,j-1,l)=ce(i,j);
                            end
                           c(i,j,0)-dax(4)/u(4)*partial(c(i,j,0),distancia)
                           =ce(i,j);
                         end
                    end
                end
            end
        end
    end
end

EQUATION
#u(j)
u(1)=u(4)+ue;
u(2)=u(1)-ux;
u(3)=u(2)+ufeed;
u(4)=u(3)-ur;

#dax:=(u*l)/pe
for j:=1 to ns do
    dax(j)=u(j)*l/pe ;
end

for j:=1 to nc do
    if (j>=y1 and j<y1+x(1)) or j<y1+x(1)-nc  then
       for i:=1 to n do
           for k:=0|+ to l|- do
              $c(i,j,k)=dax(1)*partial(c(i,j,k),distancia,distancia)
              -u(1)*partial(c(i,j,k),distancia)
              -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
           end
       end
    else
        if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
            for i:=1 to n do
                for k:=0|+ to l|- do
                    $c(i,j,k)=dax(2)*partial(c(i,j,k),distancia,distancia)
                    -u(2)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
            end
         else
            if (j>=y3  and j<y3+x(3)) or j<y3+x(3)-nc then
                 for i:=1 to n do
                    for k:=0|+ to l|- do
                       $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                        -u(3)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                    end
                end
            else
                #if (j>=y4 and j<y4+x(4)) or j<y4+x(4)-nc then
                for i:=1 to n do
                    for k:=0|+ to l|- do
                       $c(i,j,k)=dax(4)*partial(c(i,j,k),distancia,distancia)
                       -u(4)*partial(c(i,j,k),distancia)
                       -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
```

```
                    end
                end
            end
        end
    end
end

for j:=1 to nc do
    for i:=1 to n do
        for k:=0 to 1 do
            $q(i,j,k)=kl(i)*(qq(i,j,k)-q(i,j,k));
        end
    end
end

for j:=1 to nc do
        for k:=0 to 1 do
            qq(1,j,k)=2.69*c(1,j,k)/(1+0.0336*c(1,j,k)
            +0.0466*c(2,j,k))+0.10*c(1,j,k)/(1+c(1,j,k)+3*c(2,j,k));
            qq(2,j,k)=3.73*c(2,j,k)/(1+0.0336*c(1,j,k)
            +0.0466*c(2,j,k))+0.30*c(2,j,k)/(1+c(1,j,k)+3*c(2,j,k));
        end
end

if y1+x(1)<=nc+1 then aa=y1+x(1)-1; else  aa=y1-(nc-(x(1)-1)); end
if y3+x(3)<=nc+1 then bb=y3+x(3)-1; else  bb=y3-(nc-(x(3)-1)); end
for j:=1 to nc do
    if j=aa then
       $cx1(j)=c(1,j,l)/tt;#Qx*c(1,j,l)/tt;
       $cx2(j)=c(2,j,l)/tt;#Qx*c(2,j,l)/tt;
       $cr1(j)=0;
       $cr2(j)=0;
    else
        if j=bb then
           $cr1(j)=c(1,j,l)/tt;#ur*A*epsilon*c(1,j,l)/tt;
           $cr2(j)=c(2,j,l)/tt;#ur*A*epsilon*c(2,j,l)/tt;
           $cx1(j)=0;
           $cx2(j)=0;
        else
                $cx1(j)=0;
                $cx2(j)=0;
                $cr1(j)=0;
                $cr2(j)=0;
        end
    end
end


u(4)=Q4/A/epsilon ;
ue=Qe/A/epsilon ;
ux=Qx/A/epsilon ;
ufeed=Qfeed/A/epsilon;

ASSIGN
contador1:=1;
contador:=1;
Qe:=27.4e-3;
Qx:=24.1e-3;
Q4:=35.7e-3;
Qfeed:=5.0e-3;
y1:=1;
y2:=1+x(1);
y3:=1+sigma(x(1:2));
```

```
y4:=1+sigma(x(1:3));
pr:=0;
px:=0;
recr:=0;
recx:=0;
prod:=0;
EC:=0;

INITIAL
    for j:=1 to nc do
        cx1(j)=0;
        cx2(j)=0;
        cr1(j)=0;
        cr2(j)=0;
        for i:=1 to n do
            for k:=0|+ to l|- do
                c(i,j,k)=0;
            end
        end
    end

   for j:=1 to nc do
        for i:=1 to n do
            for k:=0 to l do
                q(i,j,k)=0;
             end
        end
    end
```

B.4 Varicol model

```
#                                VARICOL PROCESS
# Needed programs
# functionVARICOL

UNIT
VARICOL as functionVARICOL

SET
within VARICOL do
    tt:=2.18;
    stepc:=1;
    xx:=[1,3,2,3];
    fator:=15;
    nt:=fator*sigma(x(1:4));
    nc:=sigma(xx(1:4));
    kl(1):=6;
    kl(2):=6;
    T:=303.15;
    pe:=2000 ;
    epsilon:=0.4;
    d:=0.26;
    n:=2 ;
    nj:=4;
    l:=2.1/x(1);
    A:=atan(1)*4*(d/2)^2 ;
    cfeed(1):=2.9;
    cfeed(2):=2.9;
    distancia:=[OCFEM,2,300];
end
```

```
SOLUTIONPARAMETERS
ReportingInterval := 0.01

SCHEDULE
while VARICOL.contador<= VARICOL.nt do
      sequence
            continue for VARICOL.tt*0.5
                 if VARICOL.y4< VARICOL.nc then
                    reassign
                        VARICOL.y4:=old(VARICOL.y4)+ VARICOL.stepc;
                    end
                 else
                    reassign
                        VARICOL.y4:=1;
                    end
                 end
                 if VARICOL.y2< VARICOL.nc then
                    reassign
                        VARICOL.y2:=old(VARICOL.y2)+ VARICOL.stepc;
                    end
                 else
                    reassign
                        VARICOL.y2:=1;
                    end
                 end
              end

              reassign
                   VARICOL.x(1):=2;
                   VARICOL.x(2):=2;
                   VARICOL.x(3):=3;
                   VARICOL.x(4):=2;
              end
            continue for VARICOL.tt*0.5
                 if VARICOL.y1< VARICOL.nc then
                    reassign
                        VARICOL.y1:=old(VARICOL.y1)+ VARICOL.stepc;
                    end
                 else
                    reassign
                        VARICOL.y1:=1;
                    end
                 end
                 if VARICOL.y3< VARICOL.nc then
                    reassign
                        VARICOL.y3:=old(VARICOL.y3)+ VARICOL.stepc;
                    end
                 else
                    reassign
                        VARICOL.y3:=1;
                    end
                 end
              end

              reassign
                   VARICOL.x(1):=1;
                   VARICOL.x(2):=3;
                   VARICOL.x(3):=2;
                   VARICOL.x(4):=1;
              end
              reassign
                    VARICOL.pr:=sigma(old(VARICOL.cr1))
```

```
                    /(sigma(old(VARICOL.cr1))+sigma(old(VARICOL.cr2)));
                    VARICOL.px:=sigma(old(VARICOL.cx2))
                    /(sigma(old(VARICOL.cx1))+sigma(old(VARICOL.cx2)));
                    VARICOL.recr:=old(VARICOL.ur)* VARICOL.A* VARICOL.epsilon
                    *sigma(old(VARICOL.cr1))
                    /(old(VARICOL.Qfeed)*old(VARICOL.cfeed(1)));
                    VARICOL.recx:=old(VARICOL.Qx)*sigma(old(VARICOL.cx2))
                    /(old(VARICOL.Qfeed)*VARICOL.cfeed(2));
                    VARICOL.prod:=(old(VARICOL.recr)*old(VARICOL.ur)
                    * VARICOL.A* VARICOL.epsilon
                    *sigma(old(VARICOL.cr1)))
                    /((1- VARICOL.epsilon)* VARICOL.A* VARICOL.l* VARICOL.nc)
                     +(old(VARICOL.recx)*old(VARICOL.Qx)
                    *sigma(old(VARICOL.cx2)))/
                    ((1- VARICOL.epsilon)* VARICOL.A* VARICOL.l* VARICOL.nc);
                    VARICOL.EC:=(old(VARICOL.Qe)+old(S VARICOL.Qfeed))
                    /(old(VARICOL.Qfeed)*sigma(old(VARICOL.cfeed)));
                    VARICOL.ccr1:=sigma(old(VARICOL.cr1))/VARICOL.tt);
                    VARICOL.ccx2:=sigma(old(VARICOL.cx2))/VARICOL.tt);
                    VARICOL.ccr2:=sigma(old(VARICOL.cr2))/VARICOL.tt);
                    VARICOL.ccx1:=sigma(old(VARICOL.cx1))/VARICOL.tt);
                else
                    reassign
                       VARICOL.contador:=old(VARICOL.contador)+1;
                       VARICOL.pr:=0;
                       VARICOL.px:=0;
                       VARICOL.recr:=0;
                       VARICOL.recx:=0;
                       VARICOL.prod:=0;
                       VARICOL.ccr2:=0;
                       VARICOL.ccr1:=0;
                       VARICOL.ccx2:=0;
                       VARICOL.ccx1:=0;
                       VARICOL.EC:=0;
                   End
                end

                reinitial
                    VARICOL.cx1,
                    VARICOL.cx2,
                    VARICOL.cr1,
                    VARICOL.cr2
                with
                    VARICOL.cx1=0;
                    VARICOL.cx2=0;
                    VARICOL.cr1=0;
                    VARICOL.cr2=0;
                end
        end
end


#                               functionVARICOL
#

PARAMETER
fator as integer
stepc as integer
nt as integer
nc as integer
x as array(ns) of integer
tt as real
A as real
```

```
l as real
n as integer
nj as integer
d as real
epsilon as real
T as real
pe as real
cfeed as array(n) of real
kl as array(n) of real
xx as array(n) of real

DISTRIBUTION_DOMAIN
distancia as [0:l]

VARIABLE
Qfeed as valor
Qs as valor
Qe as valor
Qx as valor
Q4 as valor
pr as valor
px as valor
recr as valor
recx as valor
prodr as valor
prodx as valor
E as valor
P as valor
ur as valor
ue as valor
ux as valor
ufeed as valor
us as valor
y1 as valor
y2 as valor
y3 as valor
y4 as valor
u as array(nc) of valor
l as array(nc) of valor
dax as array(nc) of valor
contador as valor
aa as valor
bb as valor
cx1 as array (ns) of valor
cx2 as array (ns) of valor
cr1 as array (ns) of valor
cr2 as array (ns) of valor
c as distribution(n,nc,distancia) of valor
q as distribution(n,nc,distancia) of valor
qq as distribution(n,nc,distancia) of valor
ce as distribution (n,nc) of valor


BOUNDARY
#calculation of ce(i,j)
#Z of the first column in the section = L
for j:=1 to nc do
     for i:=1 to n do
         partial(c(i,j,l),distancia)=0;
     end
end

if y1=y2 then
```

```
for j:=1 to nc do
    if j=y2 then
        for i:=1 to n do
            if j-1<1 then
               c(i,nc,l)*u(4)/u(2)- ux/u(2)*c(i,nc,l)=ce(i,j);
            else
               c(i,j-1,l)*u(4)/u(2)- ux/u(2)*c(i, j-1,l)=ce(i,j);
            end
            c(i,j,0)-dax(2)/u(2)*partial(c(i,j,0),distancia)=ce(i,j);
        end
     else
        if (j>y2 and j<y2+x(2)) or j<y2+x(2)-nc then
           for i:=1 to n do
               if j-1<1 then
                  c(i,nc,l)=ce(i,j);
               else
                  c(i,j-1,l)=ce(i,j);
               end
               c(i,j,0)-dax(1)/u(1)*partial(c(i,j,0),distancia)=ce(i,j);
         end
      else
          if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
              for i:=1 to n do
                  if j-1<1 then
                      c(i,nc,l)=ce(i,j);
                  else
                      c(i,j-1,l)=ce(i,j);
                  end
                  c(i,j,0)-dax(2)/u(2)*partial(c(i,j,0),distancia)=ce(i,j);
                  end
          else
              if j=y3 then
                  for i:=1 to n do
                      if j-1<1 then
                          c(i,nc,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                      else
                          c(i,j-1,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                      end
                      c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                      =ce(i,j);
                   end
              else
                  if (j>y3 and j<y3+x(3)) or j<y3+x(3)-nc  then
                      for i:=1 to n do
                          if j-1<1 then
                              c(i,nc,l)=ce(i,j);
                          else
                              c(i,j-1,l)=ce(i,j);
                          end
                       c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                       =ce(i,j);
                       end
                   else
                      if (j>=y4 and j<y4+x(3)) or j<y4+x(4)-nc  then
                         for i:=1 to n do
                             if j-1<1 then
                                c(i,nc,l)=ce(i,j);
                             else
                                c(i,j-1,l)=ce(i,j);
                             end
                             c(i,j,0)-dax(4)/u(4)*
                             partial(c(i,j,0),distancia)
                             =ce(i,j);
```

```
                        end
                      else
                           for i:=1 to n do
                              if j-1<1 then
                                 0=ce(i,j);
                              else
                                 0=ce(i,j);
                              end
                             c(i,j,0)= 0;
                           end
                       end
                   end
               end
           end
       end
else
    if y2=y3 then
       for j:=1 to nc do
           if j=y1 then
              for i:=1 to n do
                  if j-1<1 then
                     c(i,nc,l)*u(1)/u(4)*ce(i,j);
                  else
                     c(i,j-1,l)*u(1)/u(4)*ce(i,j);
                  end
                     c(i,j,0)-dax(1)/u(1)
                     *partial(c(i,j,0),distancia)=ce(i,j);
              end
           else
               if (j>y1 and j<y1+x(1)) or j<y1+x(1)-nc then
                  for i:=1 to n do
                      if j-1<1 then
                         c(i,nc,l)=ce(i,j);
                      else
                         c(i,j-1,l)=ce(i,j);
                      end
                      c(i,j,0)-dax(1)/u(1)
                      *partial(c(i,j,0),distancia)=ce(i,j);
                  end
                else
                    if j=y3 then
                       for i:=1 to n do
                           if j-1<1 then
                              ufeed/u(3)*cfeed(i)-ux/u(3)
                              *c(i,nc,l)+u(1)/u(3)*c(i,nc,l)=ce(i,j));
                           else
                              ufeed/u(3)*cfeed(i)-ux/u(3)
                              *c(i,j-1,l)+u(1)/u(3)*c(i,j-1,l)=ce(i,j));
                           end
                           c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                           =ce(i,j);
                        end
                    else
                      if (j>y3 and j<y3+x(3)) or j<y3+x(3)-nc  then
                        for i:=1 to n do
                            if j-1<1 then
                                c(i,nc,l)=ce(i,j);
                            else
                                c(i,j-1,l)=ce(i,j);
                            end
                         c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                         =ce(i,j);
```

```
                        end
                     else
                       if (j>=y4 and j<y4+x(3)) or j<y4+x(4)-nc  then
                          for i:=1 to n do
                              if j-1<1 then
                                 c(i,nc,l)=ce(i,j);
                              else
                                 c(i,j-1,l)=ce(i,j);
                              end
                              c(i,j,0)-dax(4)/u(4)*
                              partial(c(i,j,0),distancia)
                              =ce(i,j);
                          end
                       else
                           for i:=1 to n do
                              if j-1<1 then
                                 0=ce(i,j);
                              else
                                 0=ce(i,j);
                              end
                              c(i,j,0)= 0;
                           end
                       end
                    end
                 end
              end
           end
        end
else
    if y3=y4 then
       for j:=1 to nc do
           if j=y1 then
              for i:=1 to n do
                  if j-1<1 then
                     c(i,nc,l)*u(1)/u(4)*ce(i,j);
                  else
                     c(i,j-1,l)*u(1)/u(4)*ce(i,j);
                  end
                     c(i,j,0)-dax(1)/u(1)
                     *partial(c(i,j,0),distancia)=ce(i,j);
              end
           else
              if (j>y1 and j<y1+x(1)) or j<y1+x(1)-nc then
                 for i:=1 to n do
                     if j-1<1 then
                        c(i,nc,l)=ce(i,j);
                     else
                        c(i,j-1,l)=ce(i,j);
                     end
                     c(i,j,0)-dax(1)/u(1)
                     *partial(c(i,j,0),distancia)=ce(i,j);
                 end
              else
                 if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
                    for i:=1 to n do
                        if j-1<1 then
                           c(i,nc,l)=ce(i,j);
                        else
                           c(i,j-1,l)=ce(i,j);
                        end
                        c(i,j,0)-dax(2)/u(2)
                        *partial(c(i,j,0),distancia)=ce(i,j);
                    end
```

```
                else
                   if j=y4 then
                       for i:=1 to n do
                           if j-1<1 then
                             ufeed/u(4)*cfeed(i)-ur/u(4)
                             *c(i,nc,l)+u(2)/u(4)*c(i,nc,l)=ce(i,j));
                           else
                              ufeed/u(4)*cfeed(i)-ur/u(4)
                              *c(i,j-1,l)+u(2)/u(4)*c(i,j-1,l)=ce(i,j));

                           end
                        c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                        =ce(i,j);
                        end
                     else
                       if (j>y4 and j<y4+x(3)) or j<y4+x(4)-nc  then
                          for i:=1 to n do
                              if j-1<1 then
                                 c(i,nc,l)=ce(i,j);
                              else
                                 c(i,j-1,l)=ce(i,j);
                              end
                              c(i,j,0)-dax(4)/u(4)*
                              partial(c(i,j,0),distancia)
                              =ce(i,j);
                          end
                        else
                            for i:=1 to n do
                               if j-1<1 then
                                  0=ce(i,j);
                               else
                                  0=ce(i,j);
                               end
                              c(i,j,0)= 0;
                            end
                        end
                     end
                end
             end
          end
       end
  else
    if y4=y1 then
       for j:=1 to nc do
           if j=y1 then
              for i:=1 to n do
                  if j-1<1 then
                     u(3)/u(1)*c(i,nc,l)-ur/u(1)*c(i,nc,l)=ce(i,j);
                  else
                    u(3)/u(1)*c(i,j-1,l)-ur/u(1)*c(i,j-1,l)=ce(i,j);
                  end
                    c(i,j,0)-dax(1)/u(1)*partial(c(i,j,0),distancia)=ce(i,j);
              end
           else
              if (j>y1 and j<y1+x(1)) or j<y1+x(1)-nc then
                 for i:=1 to n do
                     if j-1<1 then
                        c(i,nc,l)=ce(i,j);
                     else
                        c(i,j-1,l)=ce(i,j);
                     end
                     c(i,j,0)-dax(1)/u(1)
                     *partial(c(i,j,0),distancia)=ce(i,j);
```

```
                end
            else
                if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
                    for i:=1 to n do
                        if j-1<1 then
                            c(i,nc,l)=ce(i,j);
                        else
                            c(i,j-1,l)=ce(i,j);
                        end
                        c(i,j,0)-dax(2)/u(2)
                        *partial(c(i,j,0),distancia)=ce(i,j);
                    end
                else
                    if j=y3 then
                       for i:=1 to n do
                           for i:=1 to n do
                        if j-1<1 then
                            c(i,nc,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                        else
                            c(i,j-1,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                        end
                        c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                        =ce(i,j);
                    else
                        if (j>y3 and j<y3+x(3)) or j<y3+x(3)-nc  then
                           for i:=1 to n do
                               if j-1<1 then
                                  c(i,nc,l)=ce(i,j);
                               else
                                  c(i,j-1,l)=ce(i,j);
                               end
                               c(i,j,0)-dax(4)/u(4)*
                               partial(c(i,j,0),distancia)
                               =ce(i,j);
                           end
                         else
                             for i:=1 to n do
                                if j-1<1 then
                                   0=ce(i,j);
                                else
                                   0=ce(i,j);
                                end
                               c(i,j,0)= 0;
                             end
                         end
                     end
                end
            end
        end
    end
else
    for j:=1 to nc do
       if j=y1 then
            if j-1<1 then
               c(i,nc,l)*u(1)/u(4)*ce(i,j);
            else
               c(i,j-1,l)*u(1)/u(4)*ce(i,j);
            end
               c(i,j,0)-dax(1)/u(1)*partial(c(i,j,0),distancia)=ce(i,j);
        end
     else
        if (j>y1 and j<y1+x(1)) or j<y1+x(1)-nc then
           for i:=1 to n do
```

```
                    if j-1<1 then
                       c(i,nc,l)=ce(i,j);
                    else
                       c(i,j-1,l)=ce(i,j);
                    end
                    c(i,j,0)-dax(1)/u(1)
                    *partial(c(i,j,0),distancia)=ce(i,j);
                end
             else
                if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
                   for i:=1 to n do
                       if j-1<1 then
                          c(i,nc,l)=ce(i,j);
                       else
                          c(i,j-1,l)=ce(i,j);
                       end
                       c(i,j,0)-dax(2)/u(2)
                       *partial(c(i,j,0),distancia)=ce(i,j);
                   end
                else
                   if j=y3 then
                      for i:=1 to n do
                          for i:=1 to n do
                       if j-1<1 then
                           c(i,nc,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                       else
                           c(i,j-1,l)=u(3)/u(2)*ce(i,j)-ufeed/u(2)*cfeed(i);
                       end
                       c(i,j,0)-dax(3)/u(3)*partial(c(i,j,0),distancia)
                       =ce(i,j);
                   else
                       if (j>y3 and j<y3+x(3)) or j<y3+x(3)-nc  then
                          for i:=1 to n do
                              if j-1<1 then
                                 c(i,nc,l)=ce(i,j);
                              else
                                 c(i,j-1,l)=ce(i,j);
                              end
                              c(i,j,0)-dax(4)/u(4)*
                              partial(c(i,j,0),distancia)
                              =ce(i,j);
                          end
                         else
                          #if (j>=y4 and j<y4+x(4)) or j<y4+x(4)-nc then
                            for i:=1 to n do
                              if j-1<1 then
                                 c(i,nc,l)=ce(i,j);
                              else
                                 c(i,j-1,l)=ce(i,j);
                              end
                              c(i,j,0)-dax(4)/u(4)*
                              partial(c(i,j,0),distancia)
                              =ce(i,j);
                            end
                          end
                       end
                     end
                  end
               end
            end
         end
      end
   end
```

```
end

EQUATION
#u(j)
if y1=y2 then
   ue+u(4)-ux=u(2);
   u(2)=u(1)-ux;
   u(3)=u(2)+ufeed;
   u(4)=u(3)-ur;
else
    if y2=y3 then
       u(1)=u(4);
       ufeed+u(1)-ux=u(3);
       u(3)=u(2)+ufeed;
       u(4)=u(3)-ur;
     else
        if y3=y4 then
           u(1)=u(4)+ue;
           u(2)=u(1)-ux;
           ufeed+u(2)-ur=u(4);
           u(4)=u(3)-ur;
        else
           if y4=y1 then
              u(1)=u(4)+ue;
              u(2)=u(1)-ux;
              u(3)=ufeed+u(2);
              ue+u(3)-ur=u(1);
           else
              u(1)=u(4)+ue;
              u(2)=u(1)-ux;
              u(3)=ufeed+u(2);
              u(4)=u(3)-ur;
           end
       end
   end
end

#dax:=(u*l)/pe
for j:=1 to ns do
    dax(j)=u(j)*l/pe ;
end

if y1=y2 then
   for j:=1 to nc do
       if (j>=y2 and j<y2+x(2)) or j<y2+x(2)-nc then
           for i:=1 to n do
               for k:=0|+ to l|- do
                    $c(i,j,k)=dax(2)*partial(c(i,j,k),distancia,distancia)
                    -u(2)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
            end
         else
           if (j>=y3  and j<y3+x(3)) or j<y3+x(3)-nc then
                for i:=1 to n do
                   for k:=0|+ to l|- do
                      $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                       -u(3)*partial(c(i,j,k),distancia)
                       -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                   end
               end
           else
               if (j>=y4  and j<y4+x(4)) or j<y4+x(4)-nc then
```

```
                for i:=1 to n do
                  for k:=0|+ to l|- do
                     $c(i,j,k)=dax(4)*partial(c(i,j,k),distancia,distancia)
                     -u(4)*partial(c(i,j,k),distancia)
                     -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                   end
                end
             else
                for i:=1 to n do
                   for k:=0|+ to l|- do
                      $c(i,j,k)=0
                   end
                end
             end
          end
       end
    end
else
  if y2=y3 then
     for j:=1 to nc do
       if (j>=y1 and j<y1+x(1)) or j<y1+x(1)-nc then
           for i:=1 to n do
               for k:=0|+ to l|- do
                    $c(i,j,k)=dax(1)*partial(c(i,j,k),distancia,distancia)
                    -u(1)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
           end
        else
           if (j>=y3  and j<y3+x(3)) or j<y3+x(3)-nc then
                for i:=1 to n do
                   for k:=0|+ to l|- do
                      $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                       -u(3)*partial(c(i,j,k),distancia)
                       -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                   end
                end
           else
               if (j>=y4  and j<y4+x(4)) or j<y4+x(4)-nc then
                 for i:=1 to n do
                   for k:=0|+ to l|- do
                      $c(i,j,k)=dax(4)*partial(c(i,j,k),distancia,distancia)
                      -u(4)*partial(c(i,j,k),distancia)
                      -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                    end
                 end
              else
                 for i:=1 to n do
                    for k:=0|+ to l|- do
                       $c(i,j,k)=0
                    end
                 end
              end
           end
        end
     end
  else
    if y3=y4 then
       for j:=1 to nc do
           if (j>=y1 and j<y1+x(1)) or j<y1+x(1)-nc then
             for i:=1 to n do
               for k:=0|+ to l|- do
                    $c(i,j,k)=dax(1)*partial(c(i,j,k),distancia,distancia)
```

```
                    -u(1)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
            end
        else
            if (j>=y2  and j<y2+x(2)) or j<y2+x(2)-nc then
                for i:=1 to n do
                    for k:=0|+ to l|- do
                        $c(i,j,k)=dax(2)*partial(c(i,j,k),distancia,distancia)
                         -u(2)*partial(c(i,j,k),distancia)
                         -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                    end
                end
            else
                if (j>=y4  and j<y4+x(4)) or j<y4+x(4)-nc then
                  for i:=1 to n do
                    for k:=0|+ to l|- do
                        $c(i,j,k)=dax(4)*partial(c(i,j,k),distancia,distancia)
                        -u(4)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                     end
                  end
                else
                    for i:=1 to n do
                        for k:=0|+ to l|- do
                            $c(i,j,k)=0
                        end
                    end
                end
            end
        end
    end
else
    if y4=y1 then
        for j:=1 to nc do
          if (j>=y1 and j<y1+x(1)) or j<y1+x(1)-nc then
            for i:=1 to n do
                for k:=0|+ to l|- do
                    $c(i,j,k)=dax(1)*partial(c(i,j,k),distancia,distancia)
                    -u(1)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
            end
         else
            if (j>=y2  and j<y2+x(2)) or j<y2+x(2)-nc then
                for i:=1 to n do
                    for k:=0|+ to l|- do
                        $c(i,j,k)=dax(2)*partial(c(i,j,k),distancia,distancia)
                         -u(2)*partial(c(i,j,k),distancia)
                         -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                    end
                end
            else
              if (j>=y3  and j<y3+x(3)) or j<y3+x(3)-nc then
                for i:=1 to n do
                    for k:=0|+ to l|- do
                        $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                        -u(3)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                     end
                  end
                else
                  for i:=1 to n do
```

```
                    for k:=0|+ to l|- do
                       $c(i,j,k)=0
                    end
                 end
              end
           end
        end
     end
     else
        for j:=1 to nc do
           if (j>=y1 and j<y1+x(1)) or j<y1+x(1)-nc then
             for i:=1 to n do
                for k:=0|+ to l|- do
                    $c(i,j,k)=dax(1)*partial(c(i,j,k),distancia,distancia)
                    -u(1)*partial(c(i,j,k),distancia)
                    -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                end
             end
           else
              if (j>=y2  and j<y2+x(2)) or j<y2+x(2)-nc then
                 for i:=1 to n do
                    for k:=0|+ to l|- do
                       $c(i,j,k)=dax(2)*partial(c(i,j,k),distancia,distancia)
                        -u(2)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                    end
                 end
              else
                if (j>=y3  and j<y3+x(3)) or j<y3+x(3)-nc then
                   for i:=1 to n do
                     for k:=0|+ to l|- do
                        $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                        -u(3)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                      end
                   end
                else
                   if (j>=y4  and j<y4+x(4)) or j<y4+x(4)-nc then
                     for i:=1 to n do
                        for k:=0|+ to l|- do
                        $c(i,j,k)=dax(3)*partial(c(i,j,k),distancia,distancia)
                        -u(3)*partial(c(i,j,k),distancia)
                        -((1-epsilon)/epsilon)*kl(i)*(qq(i,j,k)-q(i,j,k));
                       end
                     end
                   else
                     for i:=1 to n do
                        for k:=0|+ to l|- do
                           $c(i,j,k)=0
                        end
                     end
                   end
                 end
              end
           end
        end
      end
    end
  end
end

for j:=1 to nc do
    for i:=1 to n do
```

```
        for k:=0 to l do
            $q(i,j,k)=kl(i)*(qq(i,j,k)-q(i,j,k));
        end
    end
end

for j:=1 to nc do
        for k:=0 to l do
            qq(1,j,k)=2.69*c(1,j,k)/(1+0.0336*c(1,j,k)
            +0.0466*c(2,j,k))+0.10*c(1,j,k)/(1+c(1,j,k)+3*c(2,j,k));
            qq(2,j,k)=3.73*c(2,j,k)/(1+0.0336*c(1,j,k)
            +0.0466*c(2,j,k))+0.30*c(2,j,k)/(1+c(1,j,k)+3*c(2,j,k));
        end
end

if y1+x(1)<=nc+1 then aa=y1+x(1)-1; else  aa=y1-(nc-(x(1)-1)); end
if y3+x(3)<=nc+1 then bb=y3+x(3)-1; else  bb=y3-(nc-(x(3)-1)); end
if y2+x(2)<=nc+1 then cc=y2+x(2)-1; else  cc=y2-(nc-(x(2)-1)); end
if y4+x(4)<=nc+1 then dd=y4+x(4)-1; else  dd=y4-(nc-(x(4)-1)); end

if y1=y2 then
   for j:=1 to nc do
       if j=dd then
          $cx1(j)=c(1,j,l);
          $cx2(j)=c(2,j,l);
          $cr1(j)=0;
          $cr2(j)=0;
       else
          if j=bb then
             $cr1(j)=c(1,j,l);
             $cr2(j)=c(2,j,l);
             $cx1(j)=0;
             $cx2(j)=0;
           else
             $cx1(j)=0;
             $cx2(j)=0;
             $cr1(j)=0;
             cr2(j)=0;
           end
        end
    end
else
   if y3=y4 then
      for j:=1 to nc do
          if j=aa then
             $cx1(j)=c(1,j,l);
             $cx2(j)=c(2,j,l);
             $cr1(j)=0;
             $cr2(j)=0;
          else
             if j=cc then
                $cr1(j)=c(1,j,l);
                $cr2(j)=c(2,j,l);
                $cx1(j)=0;
                $cx2(j)=0;
             else
                $cx1(j)=0;
                $cx2(j)=0;
                $cr1(j)=0;
                cr2(j)=0;
             end
          end
       end
```

```
   else
      if j=aa then
          $cx1(j)=c(1,j,l);
          $cx2(j)=c(2,j,l);
          $cr1(j)=0;
          $cr2(j)=0;
       else
          if j=bb then
             $cr1(j)=c(1,j,l);
             $cr2(j)=c(2,j,l);
             $cx1(j)=0;
             $cx2(j)=0;
           else
             $cx1(j)=0;
             $cx2(j)=0;
             $cr1(j)=0;
             cr2(j)=0;
             end
          end
       end
    end
end

u(4)=Q4/A/epsilon ;
ue=Qe/A/epsilon ;
ux=Qx/A/epsilon ;
ufeed=Qfeed/A/epsilon;

ASSIGN
contador1:=1;
contador:=1;
Qe:=27.4e-3;
Qx:=24.1e-3;
Q4:=35.7e-3;
Qfeed:=5.0e-3;
y1:=1;
y2:=1+xx(1);
y3:=1+sigma(xx(1:2));
y4:=1+sigma(xx(1:3));
pr:=0;
px:=0;
recr:=0;
recx:=0;
prod:=0;
EC:=0;
ccr2:=0;
ccr1:=0;
ccx1:=0;
ccx2:=0;
x(1):=1;
x(2):=3;
x(3):=2;
x(4):=1;

INITIAL
    for j:=1 to nc do
        cx1(j)=0;
        cx2(j)=0;
        cr1(j)=0;
        cr2(j)=0;
        for i:=1 to n do
            for k:=0|+ to l|- do
                c(i,j,k)=0;
```

```
            end
        end
    end

for j:=1 to nc do
        for i:=1 to n do
            for k:=0 to l do
                q(i,j,k)=0;
             end
        end
    end
```

Appendix C

W analysis

From the study in subchapter 5.1.2, besides the original equation proposed by Shi and Eberhart (1998), Equations 5.12 and 5.14 were tested to know the behavior of the system in the presence of *w* with total dispersion and with a trend dispersion, respectively. Since a constant *w* did not lead to acceptable results, Equation 5.13 was not tested.

First, the simulations were performed, using the parameters of Table C.1 and the objective function of Equation 5.8 (see section 5.1.1). The results are in Tables C.2 and C.3.

Table C.1 Optimization C.1 and *C.2* parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	0.01	200
Q_s	0.01	200
n_{it}	500	
n_d	5	
n_p	50	
ω	105	
P^{set}	0.97	
factor	2	

Table C.2 Optimization C.1 (total dispersion *w*) results. (flow-rates in mL/min, productivity in g/L_{ads}/day and CPU time in hours).

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	*fobj*×10^2	CPU
Run1	**	168.5	*	6.8	12.2	97.0	97.0	206.9	87.5	3.8
Run2	**	171.6	3.3	6.8	12.2	97.0	97.0	207.1	87.5	3.5
Run3	192.4	160.3	*	6.8	12.4	97.0	97.0	207.0	87.5	3.4
Run4	190.8	186.0	27.6	7.0	12.5	97.0	97.0	211.8	87.3	3.7
Run5	75.7	61.6	17.8	6.9	12.4	97.0	97.0	210.0	87.4	3.4
Run6	131.2	123.2	24.9	6.9	12.7	97.0	97.0	211.1	87.3	3.6
Run7	**	177.1	8.8	6.8	12.3	97.0	97.0	207.7	87.5	3.4
Run8	187.3	170.9	15.1	6.9	12.3	97.0	97.0	208.6	87.4	3.3
Run9	157.6	130.9	4.8	6.8	12.2	97.0	97.5	207.1	87.5	3.5
Run10	80.6	53.8	5.4	6.8	12.5	97.0	97.2	207.5	87.4	3.3

* minimum was attained (0.01 mL/min)
** maximum was attained (200 mL/min)

Table C.3 Optimization C.2 (trend dispersion *w*) results. (flow-rates in mL/min, productivity in g/L_{ads}/day and CPU time in hours).The blue shadow represents an outlier.

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	*Prod*	$fobj\times10^2$	CPU
Run1	165.1	151.3	19.4	6.9	12.7	97.0	97.0	209.1	87.4	3.3
Run2	183.5	178.2	27.5	7.0	12.6	97.0	97.0	211.7	87.3	3.2
Run3	61.3	36.3	7.3	6.9	12.5	97.0	97.0	209.1	87.4	2.9
Run4	160.5	149.9	21.3	6.9	12.3	97.0	97.0	209.9	87.4	3.2
Run5	199.3	125.8	*	1.2	22.8	116.6	97.4	38.0	97.4	3.0
Run6	119.5	80.8	41.8	*	24.6	320.5	99.4	21.6	98.5	3.6
Run7	196.5	187.4	23.0	6.9	12.4	97.0	97.0	210.3	87.3	3.0
Run8	108.5	103.8	27.5	7.0	12.5	97.0	97.0	212.0	87.2	3.1
Run9	**	194.7	28.5	6.9	13.0	97.0	97.5	211.3	87.3	3.2
Run10	93.0	75.0	14.3	6.9	12.5	97.0	97.2	208.9	87.4	3.9

* minimum was attained (0.01 mL/min)
** maximum was attained (200 mL/min)

Comparing these results with the ones in section 5.1.2 (O*ptimization 5.3*, Table 5.4), the feed and the solid flow-rates have similar average values. However, the trend dispersion *w* show two outliers instead of one; unlike this one, the total dispersion *w* does not have outliers.
In section 5.3, a more complex objective function (Equation 5.23) is presented. Once the PSO in very sensitive to the parameters, the *w* test was also perfomed with section's 5.3 conditions (Table C.4). The results are in Tables C.5 and C.6.

Table C.4 Optimization C.3 and *C.4* parameters (flow-rates in mL/min).

	min	*max*
Q_E	0.01	200
Q_X	0.01	200
Q_{IV}	0.01	200
Q_{feed}	0.01	200
Q_s	0.01	200
n_{it}	2000	
n_d	5	
n_p	50	
ω	4000	
P^{set}	0.97	
factor	5	

Table C.5 Optimization C.3 (total dispersion *w*) results (flow-rates in mL/min, productivity in g/L_{ads}/day, eluent consumption in dL/g and CPU time in hours).The blue shadow represents an outlier.

	Q_e	Q_x	Q_{IV}	Q_{feed}	Q_s	P_r	P_x	*Prod*	$fobj \times 10^2$	*EC*	CPU
Run1	39.6	33.9	28.0	7.1	13.0	97.0	97.0	215.0	747.9	113.6	7.3
Run2	112.0	104.8	26.0	7.0	12.8	97.0	96.7	213.5	937.5	292.3	6.3
Run3	36.2	30.4	24.6	7.0	11.9	96.8	97.0	212.9	747.7	106.0	8.4
Run4	27.5	24.2	27.6	7.0	12.1	97.0	97.0	212.5	725.5	78.0	7.4
Run5	27.1	23.8	27.4	7.0	12.0	97.0	97.0	212.0	725.4	84.1	7.9
Run6	27.2	23.9	27.4	7.0	12.0	97.0	97.0	212.1	725.4	84.3	7.8
Run7	27.1	23.8	27.4	7.0	12.0	97.0	97.0	212.0	725.4	84.1	7.8
Run8	52.5	23.6	*	6.8	11.4	97.0	96.8	206.4	807.9	149.9	7.6
Run9	27.2	23.9	27.4	7.0	12.1	97.0	97.0	212.1	725.4	84.3	12.1
Run10	52.5	23.6	*	6.8	11.4	97.0	96.8	206.4	807.9	149.9	7.5

* minimum was attained (0.01 mL/min)

Table C.6 Optimization C.4 (trend dispersion *w*) results (flow-rates in mL/min, productivity in g/L_{ads}/day, eluent consumption in dL/g and CPU time in hours)-.

	Q_e	Q_x	Q_{IV}	Q_{feed}	Q_s	P_r	P_x	*Prod*	$fobj \times 10^2$	*EC*	CPU
Run1	49.3	22.0	*	6.7	10.9	97.0	97.0	203.2	810.2	143.6	8.0
Run2	105.4	100.9	28.0	7.1	12.6	97.0	96.7	215.5	919.8	273.7	8.9
Run3	88.6	84.2	27.9	7.0	12.6	97.0	97.0	212.9	873.5	235.7	8.9
Run4	52.6	47.8	28.0	7.1	12.7	97.0	97.0	214.4	718.6	145.7	8.8
Run5	56.8	26.0	*	6.9	12.3	97.0	96.8	209.6	811.1	158.3	7.8
Run6	30.8	26.8	30.7	7.0	13.3	97.0	97.0	212.0	737.9	93.3	7.8
Run7	84.2	42.3	67.9	*	34.1	1.0	97.0	18.5	7492.5	3109.5	10.6
Run8	56.4	26.1	*	6.9	11.9	97.0	96.8	209.7	810.8	157.2	7.8
Run9	34.8	28.6	25.2	6.8	12.2	97.0	97.1	208.1	752.5	104.9	8.4
Run10	40.4	36.1	29.0	7.1	12.9	97.0	97.0	215.7	750.2	115.3	7.6

* minimum was attained (0.01 mL/min)

Altough in optimizations C.1 and C.2 the total and the trend *w* led to similar results, comparing with the ones obtained, using Shi and Eberhart's equation (despite the outliers); in optimizations C.3 and C.4, as the complexity of the objective function increases, neither the total or the trend dispersion *w* led to better results. The total dispersion *w* can be considered as an acceptable result (when comparing with Table 5.8), but it is clear that the number of outliers is higher. The trend dispersion *w* is not suited for this system; in fact, the results present a large dispersion.

From the results shown in this appendix, it is visible that using the equation proposed by Shi and Eberhart (1998) was a good choice to optimize this system with the PSO algorithm.

Appendix D

Complementary analysis

To check if the alteration on the calculation of x_p^{i+1} influences significantly the algorithm, *Simulation 5.1* was repeated, using the original formula

$$x_p^{i+1}=x_p^i+v^i \tag{D.1}$$

The results are in Table D.1.

Table D.1 Optimization D.1 and comparison with Wu et al. (2006),(flow-rates in mL/min, productivity in g/L_{ads}/day, CPU time in hours).

	Q_E	Q_X	Q_{IV}	Q_{feed}	Q_s	P_R	P_X	Prod	$fobj\times10^2$	CPU
Wu *et al.*	22.1	19.6	21.5	4.4	9.0	99.3	97.6	144.0	94.3	-
Run1	20.6	16.8	27.3	**	11.5	97.8	98.5	153.9	90.4	1.3
Run2	**	27.6	23.5	**	9.9	99.0	98.5	155.3	90.3	1.3
Run3	**	17.9	10.6	**	8.8	98.9	98.4	154.1	90.3	1.3
Run4	**	27.5	23.5	**	9.9	99.0	98.5	155.3	90.3	1.4
Run5	28.8	26.7	22.3	**	9.3	99.0	98.5	155.3	90.3	1.3
Run6	29.0	26.4	23.5	**	9.9	99.0	98.5	155.3	90.3	1.4
Run7	**	27.1	24.1	**	10.2	98.9	98.5	155.2	90.3	1.3
Run8	27.5	26.2	20.9	**	8.7	99.1	98.3	155.3	90.3	1.3
Run9	**	28.9	20.5	**	8.5	99.1	98.2	155.3	90.3	1.2
Run10	26.3	24.9	21.1	**	8.8	99.0	98.3	155.3	90.3	1.3

** maximum was attained (200 mL/min)

Comparing with *Optimization 5.1*, it is noticeable that the results are very similar; the difference between this optimization and the one performed by Wu (2006) is ,then, not due to the update of x_p^{i+1}. This difference might be explained by the numerical methods that were used to simulate the process model. Moreover, there is no reference of the algoritm's programming details (for example, if some constraint was used to restrain the initial family of particles, it would influence the optimization).

Although the change in the calculation of x_p^{i+1} did not improve significantly the algorithm in this optimization, the results were not worse so it was still used on this work (moreover, the PSO is a sensitive algorithm and this change might have a positive effect in other optimization cases).